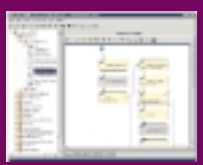
FEDERATION IN B2B WEB SERVICES

November 2002 Volume 2 Issue 11

WebServices JOURNAL

.NET J2EE XML

WSJ2.COM



WebServices JOURNAL
.NET J2EE XML

INTERNATIONAL ADVISORY BOARD
Jeremy Allaire, Andrew Astor, Steve Benfield, Graham Glass, Tyler Jewell, Paul Lipton, Norbert Mikula, Frank Moss, George Paolini, James Phillips, Simon Phipps

TECHNICAL ADVISORY BOARD
Bernhard Borges, JP Morgenthal, Andy Roberts, Ajit Sagar, Michael A. Sick, Simeon Simeonov

EDITORIAL
EDITOR-IN-CHIEF
Sean Rhody sean@sys-con.com
EDITORIAL DIRECTOR
Jeremy Geelan jeremy@sys-con.com
INDUSTRY EDITOR
Norbert Mikula norbert@sys-con.com
PRODUCT REVIEW EDITOR
Joe Mitchko joe@sys-con.com
.NET EDITOR
Dave Rader davidr@fusiontech.com
EXECUTIVE EDITOR
Gail Schultz gail@sys-con.com
MANAGING EDITOR
Cheryl Van Sise cheryl@sys-con.com
EDITORS
M'lou Pinkham mpinkham@sys-con.com
Nancy Valentine nancy@sys-con.com
ASSOCIATE EDITORS
Jamie Matusow jamie@sys-con.com
Jean Cassidy jean@sys-con.com
ASSISTANT EDITOR
Jennifer Stilley jennifer@sys-con.com

PRODUCTION
PRODUCTION CONSULTANT
Jim Morgan jim@sys-con.com
LEAD DESIGNER
Richard Silverberg richards@sys-con.com
ART DIRECTOR
Alex Botero alex@sys-con.com
ASSOCIATE ART DIRECTORS
Cathryn Burak cathyb@sys-con.com
Louis Cuffari louis@sys-con.com
ASSISTANT ART DIRECTOR
Tami Beatty tami @sys-con.com

CONTRIBUTORS TO THIS ISSUE
Srinivasa Rao Aravilli, Allen Brown, John Hagel, Lakshmi Hanspal, Paul Holland , Frank Lacombe, Paul Lipton, Mark Little, Joe Mitchko, Ken Molay, Sean Rhody, Gunjan Samtani, Keith Shaffer, Doron Sherman, Karen Thure, Jim Webber

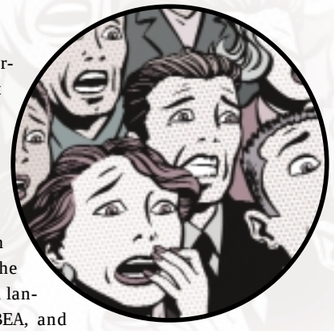
EDITORIAL OFFICES
SYS-CON MEDIA
135 CHESTNUT RIDGE ROAD, MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 FAX: 201 782-9637
WEB SERVICES JOURNAL (ISSN# 1535-6906)
Is published monthly (12 times a year)
By SYS-CON Publications, Inc.
Periodicals postage pending
Montvale, NJ 07645 and additional mailing offices
POSTMASTER: Send address changes to:
WEB SERVICES JOURNAL, SYS-CON Publications, Inc.
135 Chestnut Ridge Road, Montvale, NJ 07645

SYS-CON MEDIA

Animal Farm

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



There's a lot going on in the world of Web services these days, so much so that it's almost hard to keep track of it. And there is definitely enough to make it difficult to make sense of competing initiatives. But it certainly makes for interesting watching.

Case in point is the Web Services Interoperability organization (WS-I). Chartered as an open standards group whose purpose is to increase the interoperability of Web services across platforms, languages, and systems, the group counts IBM, BEA, and Microsoft amongst its members. So where, you ask, is Java founder and J2EE specification definer Sun Microsystems? I ask the same question. It's ironic that a group whose stated purpose is to increase interoperability cannot actually come to agreement on its membership. How can Sun, which spearheads the Java Community Process on J2EE (and on Web services) be excluded? I get the usual fingerpointing when I ask representatives of various organizations this question. To be honest, I don't care about the answer, or who's to blame. Sun should be part of the group, and that's all there is to it. The rest is just corporate posturing that serves no one. WS-I isn't complete until Sun plays.

That's not to say that the gang of three and their cohorts aren't doing good work, both in the WS-I and outside. One great point from the unlikely collaboration of these titans is the release of three specifications – BPEL4WS, WS-Coordination, and WS-Transaction. I can't take credit for the creation of these spefications myself, but ***WSJ*** has been calling for them almost since our first issue.

WS-Coordination provides a framework and protocols for the coordination of distributed applications. Whether the coordination is between groups within a single company or between companies across the Internet, this coordination of services provides a necessary component of federation of Web services, and acknowledges the fact that they may be deployed on multiple platforms and will require coordination.

The WS-Transaction specification is one of the protocols within the WS-Coordination framework that enables the creation of atomic transactions spanning multiple Web services.

And in a rare burst of cooperation not seen between two rivals since before the OS/2 days, the release of Business Process Execution Language for Web Services (BPEL4WS) unites and supercedes the efforts of Microsoft's BizTalk and IBM's Web Services Flow Language. Too bad the acronym is so darn hard to say. What this does mean is that a majority of the major players in the industry have agreed to use a single set of APIs for coordination, transaction and business process description and management. I've been on my soapbox for a year now extolling the need for these components. Fortunately, the proponents of these specifications represent a significant segment of the industry, so it's likely that adoption will occur.

But some may accuse these merry philanthropists of creating these specifications strictly for their own gain. And I might be one of those, because this space is also occupied by EAI, and no pure-play EAI vendor is represented in this specification process, although some companies, such as SeeBeyond and webMethods, are members of WS-I. Despite their presence, they weren't invited to play in the big Web services game. That's because IBM, BEA, and Microsoft have decided to divide up the Web services pie for themselves. In the words of George Orwell, "Everyone is equal, but some are more equal than others." It's time to create an independent organization for Web services, or at least empower OASIS or W3C to run the show, and make sure that other voices are heard. Not that these specifications are bad, by any means. But a monopoly of three is no different from a monopoly of one, and any monopoly is a bad thing.

Service Grids – A Missing Link in Web Services

A step toward unleashing the full potential of Web services

John Hagel

John Hagel is a business consultant based in Silicon Valley. He has served as a senior executive with several technology companies and spent 16 years as a partner at McKinsey & Company, Inc. John is the author of the forthcoming book, Out of the Box: Strategies for Achieving Profits Today and Growth Tomorrow through Web Services *(Harvard Business School Press). For more information, visit www.johnhagel.com*
JOHN@JOHNHAGEL.COM

The hype surrounding Web services mounts with every day. Dynamic composition of applications. Obsolescence of traditional applications. The claims escalate as technologists work mightily to craft compelling visions of disruptive technologies.

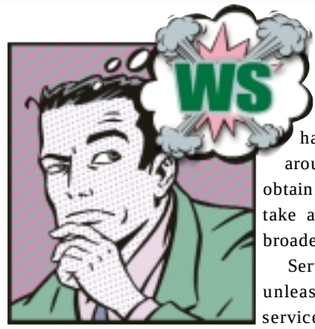
The users of technology – the business line managers who must meet quarterly budget targets – develop allergic reactions as these visions pour forth. Their skepticism stems from deepening disappointment over the limited returns generated from previous technology visions, including the rush to build Web sites and install massive enterprise applications.

These users are increasingly focused on near-term results, especially in the form of significant operating cost or asset savings. They don't want visions, they want business cases. And they are deploying Web services technology, but it's not where we might expect.

Early adopters are deploying Web services in production environments, driven by powerful economic benefits. Simply put, the technology delivers significant operating savings (not just IT savings) with relatively modest investments and very short lead times. It helps users extract even more economic value from technology investments that are already made. In this environment, that's a compelling proposition.

The technology enables lower-cost and more flexible connections across existing applications. This isn't (yet) about composing new applications, it's about exposing functionality of existing applications and making it accessible to other applications. Surprisingly, the technology is being deployed first at the edge of the enterprise, facilitating connections with business partners in key business processes like supply chain management and sales channel management. This is precisely where the complexity of connecting with very diverse technology platforms most challenges executives. It is also where this new technology has the most distinctive advantage relative to existing technologies like EAI. A diverse array of companies, from General Motors and Dell to Eastman Chemical and Wachovia, are finding real value can be generated now.

But there's a hitch. We're talking about mission-critical business processes extending beyond the firewall and encompassing many enterprises. Web services standards are helpful in facilitating connections, but more than standards are required to deliver mission critical functionality in these environments. Early adopters have creatively fashioned workarounds and accepted some risk to obtain the benefits. But it's going to take a lot more to accelerate and broaden adoption.



Service grids will be required to unleash the full potential of Web services technology. Service grids represent federations of managed enabling services designed to support more robust and flexible connections across application services. These enabling services help to ensure mission-critical functionality, including reliable messaging, security, and third-party performance auditing. These managed enabling services will be delivered by a variety of specialized utilities, operating either within an enterprise or, more likely over time, operating as independent businesses. Early examples of these utilities include VeriSign and Contivo. Large companies may organize their own service grids, knitting together specialized utilities to meet their specific needs. More generally, service grid integrators will emerge to assemble the appropriate group of specialized utilities for particular application environments.

These service grids are essential to cost effectively support connections in business processes spanning multiple enterprises. By sharing enabling services, enterprises significantly reduce the investment and technology expertise required at the connection nodes. By relying on managed services rather than simply deploying technology, enterprises will be better able to cope with the inevitable exceptions that arise when they seek to reconcile differing policies, processes, and semantics. By focusing people on managing the technology across many application environments, service grid utilities will be able to accelerate learning regarding performance requirements and develop appropriate skills and technology to push the performance envelope.

Service grids are still in a very early stage of formation. Early contenders include specialized providers like Grand Central or companies moving into this space from adjacent arenas, like E2open and Commerce One. For providers, service grids represent a significant business opportunity. For technology users, service grids represent a significant business requirement. To avoid a hitch in the take-off of Web services technology, that requirement must be met.

Written by Ken Molay

Business Rules: The Perfect Complement to Web Services

A proven paradigm

WSJ readers are already familiar with the concept and promise of Web services. For some time, media and industry analysts have been touting the revolution about to occur in the programming world as a result of universally accessible, reusable code that can be assembled to accomplish any business task.

Some of the claims are rightly looked on as hype, but there's an underlying solidity to the Web services model that makes it a compelling approach to software development. This article examines the interplay between Web services and a similar programming paradigm: business-rules management. There are many overlaps between business-rules management and Web services, but the former has been successfully implemented in far more real-world business applications. A look at the one may give enterprise developers a better idea of how to best work with the other.

Web Services

At their most fundamental level, Web services are merely programming "subroutines" that happen to be made available over the Internet. They are self-contained coding fragments that accomplish specific, well-defined tasks given a set of data to work upon. As a component business function, they do not mandate a visual interface or other I/O mechanism to communicate the results of their computations. That's left to the control of the calling application.

Of course, Web services offer a number of advantages over traditional callable code blocks. They are location independent – the code need not be physically owned, stored, or maintained by the using entity; they have a standardized access protocol; and they can be updated and repaired without affecting other parts of the application.

In their ideal realization, Web services provide a way of exposing business assets that empowers business users to specify complex business functionality in a clear, building-block fashion that can be quickly implemented and just as quickly modified.

Business Rules

Business-rules management has many of the same goals, barring the obvious aspects of Web-based availability. It's used to separate business decision processes from the mechanics of application I/O and control code. Rule services represent well-defined action determination tasks that can be called upon from many different applications. Business rules can be maintained and updated separately from the remainder of the program code. And they offer greatly improved visibility and comprehension to nontechnical business policy makers who wish to define and control business operations without worrying about programming syntax.

From an enterprise usability standpoint, business rules are best viewed not in their individual, atomic format, but as groupings intended to solve a particular decision task. Although the creation of rule services involves defining unique rules one at a time, a look at a single rule is fairly uninteresting from a business functionality outlook. One rule might validate a particular data value. Another might modify a discount rate based on a piece of customer demographic data. And a third might determine whether a product is appropriate for display in a selection list. Although each of these is an important low-level decision, none gives a sense of the overall business goal. One might refer to the rules as the "how" rather than the "what" of a business function. The "what" of a business decision process is a much more general description… "determine whether to underwrite a loan for this customer" or "respond appropriately to this employee's service request."

Business-rules management systems typically offer ways to group rules into functional blocks that accomplish a subtask. The subtasks can be combined in a procedural decision flow to conditionally use the proper subtasks in sequence to achieve the desired business decision. The overall decision flow can be viewed as a holistic rule service that is self-contained and can be accessed from any calling application when needed.

Business Rules and Web Services

In this view of rule services, it becomes clear that they are naturally suited for access using the Web services paradigm. A well-formed rule service has a clear calling point, is able to operate without further intermediary control from the calling application, and satisfies a clearly defined business function. All that's required is a way to publish the rule service using a Web services UDDI framework. The more advanced rules management software products come with automated generators to wrap rule services in the proper code constructs to enable them as Web services on the major Java application

Author Bio

Ken Molay is director of Blaze Advisor product evangelism at Fair, Isaac and Company. Ken is responsible for informing and educating companies about the implementation benefits and applications of rules-management technology across industries. He works closely with customers to understand and promote their use of rules-based business decision software for efficiency, consistency, and control of business operations.
KENMOLAY@FAIRISAAC.COM

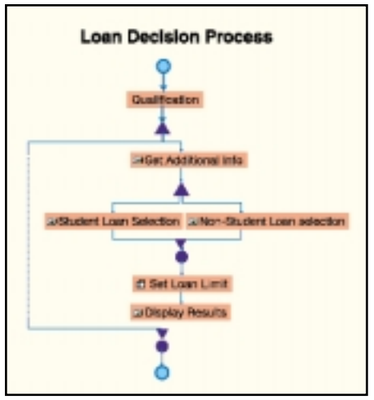


FIGURE 1 **Rule services designed to accomplish a high-level business function**

servers, ready to be published in a desired directory.

Individual rules are structured as a pairing of an explicit condition set (or rule premise) and a desired action. The most common way of thinking of this structure is the traditional IF-THEN programming construct: If a given condition is encountered, then perform the corresponding action. Naturally, rules may need to access data sources in the course of the execution. It would be inefficient and extremely difficult to pass every possible data point as an argument to the rule service on the off chance that a rule might need to examine it. So rules need on-the-fly access to external data sources as determined at run time. For instance, we might write the following rule: If Customer's Age is between 13 and 17 then set Customer's AgeGroup to TEENAGER.

The age of the customer could be looked up in a customer profile database, it could be read from an XML document, or it might be in memory as part of a Java object based on a recent transaction. Similarly, the AgeGroup value might be updated in a database or other data object as a result of executing the rule.

A more interesting situation is the case in which a rule's premise or action doesn't rely on a single property, but involves a complex function: If <Calculate Customer's Credit Rating> is greater than 1500 then <Issue Gold Card>.

The two phrases in angle brackets represent entire self-contained processes that the rule needs to initiate. These are natural candidates for Web services, since their tasks may not be carried out by the company determining that they're needed. They could be written and hosted by another party and accessed for a fee or through a licensing arrangement. A good rules-management solution plans for this type of access and allows direct access to Web services from within individual rules.

The previous two examples showed how Web services and business rules naturally work together to add value and power to component-oriented business processing. There is a third way that rules can support existing Web services, through authentication and authorization.

Security

Security of Web service processing is a major concern in making it practical for widespread use. Various companies, organizations, and standards committees are working on constructs for encryption/decryption, message confirmation, and user logon identification. Microsoft's .NET Passport and Sun's Liberty Alliance are two examples of single sign-on (SSO) frameworks that give companies "hooks" for tying their proprietary authorization strategies and policies to the external user information needed to make access decisions.

The mechanics to enable authorization and authentication are a major part of the solution. But even with these underlying building blocks in place, companies are still burdened with the task of clearly and consistently defining their rules for granting, refusing, and restricting use of systems and services. These rules are subject to change as user classifications are modified, new services are made available, and corporate policies change. A business-rules component that works with the security interfaces offers the capabilities needed in this area and allows companies to assert control over their Web services use.

UDDI

A final area of interface between Web services and business rules is UDDI (Universal Description, Discovery, and Integration). UDDI provides a means for describing published services, discovering businesses and business functions available to the caller, and integrating published services into calling applications. As with the security features mentioned above, UDDI addresses the technical infrastructure necessary to accomplish these tasks, but leaves their practical implementation "as an exercise for the reader." If Web services are to become a widespread shared service component and a potential revenue generator for publishers, enterprises will need a way to define rules for directory searches based on business-function descriptions, preferred partners, costs, and other factors that will be highly variant from company to company. The use of business-rules software will enable organizations to define and modify search and discovery criteria as needed.

What Lies Ahead

Web services are just beginning to move from an interesting theoretical model to a practical solution for companies. The first uses of Web services are likely to be intracompany, as an internal means for writing and accessing reusable business functions. Keeping Web services off publicly accessible Internet directories reduces security and discovery issues. In this limited-use scenario, Web services share many of the characteristics of business-rule systems. Each represents a self-contained business goal made up of subtasks that can be executed without intervention from the calling application. Business-rule services therefore make excellent candidates for Web services and have the advantage of being a tried and tested means of building component-oriented applications. Business rules are also ideally suited for making use of Web services to initiate complex conditions or action processes.

As companies progress to more aggressive use of public Web services shared between organizations, business rules will assist in defining and maintaining security of the shared services while offering a standardized means of searching and discovering applicable Web services based on proprietary criteria.

Companies interested in Web services can benefit from examining business-rules management as a technology to help them organize their new development work in a proven manner, while offering ways to control and manage expanded services in the future.

Straight-Through Processing and Orchestration of Web Services

A paradigm to achieve internal and external STP

Written by Gunjan Samtani and Doron Sherman

AUTHOR BIOS:

Gunjan Samtani is divisional vice president, information technology, at UBS PaineWebber, one of the world's leading financial services firms. He has several years of experience in the management, design, architecture, and implementation of large-scale EAI and B2B integration projects. Gunjan is the primary author of B2B Integration: A Practical Guide to Collaborative E-Commerce (Imperial College Press) and Web Services Architectures and Business Strategies *(Wrox Press).*
GSAMTANI@UBSPW.COM

Doron Sherman is the CTO of Collaxa, the leading provider of Web service orchestration technology. Prior to Collaxa, Doron was a chief scientist and founder of NetDynamics, the startup that pioneered the Java application server in early 1995 and led this market until its merger with Sun Microsystems in August 1998.
DORON@COLLAXA.COM

One word can describe the current state within financial organizations as far as straight-through processing (STP) is concerned: confusion.

In the current state of STP implementation in almost every financial institution, its scope is still limited and it targets only a portion of the underlying financial instruments and product and business lines, and requires a full development cycle for each product addition.

This article presents a paradigm to achieve internal and external STP through the orchestration of Web services. We discuss the fundamentals of STP, introduce the concept of orchestration, relate how business-critical STP processes can be orchestrated as Web services, and envision building the entire STP model over a service-oriented architecture (SOA)–based framework.

What Is Straight-Through Processing?

Straight-through processing, a solution that automates the end-to-end processing of transactions for all financial instruments, from initiation to resolution, is positioned to revolutionize the financial industry. STP encompasses a set of internal and external applications, business processes, and standards that will redefine the settlement and processing paradigm within the capital markets industry. It aims to make trade processing as automated as possible, allowing STP-related business processes to be carried out without unnecessary human intervention, thereby reducing to a minimum the overall processing lead time and the related risks, including inevitable human errors.

STP can be categorized into three logical levels (see Figure 1), each comprised of multiple applications and systems:

1. Transactional
2. Asset Management
3. Informational

STP Encompasses EAI and B2Bi

STP encompasses both enterprise application integration (EAI) and business-to-business application integration (B2Bi; see Figure 2). EAI for STP, also known as *internal STP*, relates to the trade and settlement processes that are internal to an industry participant. B2Bi for STP, also known as *external STP*, is about connecting seamlessly to all external partners in the trading and settlement process, including the industry-matching utilities such as GSCC's RTTM and Omgeo. The external partners include custodians, exchanges, clearing corporations, central security depositories, and other information providers.

BPM: The Most Important Component of STP

Business process management (BPM) is the most important component of STP. It enables financial enterprises to automate and integrate the disparate internal and external corporate business processesby supporting dynamic process topologies that allow the boundary between processes and participants to be determined either statically or dynamically on a real-time basis. Further, its implementation will provide every financial corporation with the opportunity to redefine and automate core business processes, resulting in streamlined business operations and reduced costs.

BPM for internal STP would enable companies to achieve internal systems that are truly integrated through the automation of information flows. The business processes that control information flow by coordinating interactions with business applications and systems within an organization are called *private* or *closed* STP processes.

BPM for external STP would focus on how financial institutions, as a vertical industry, can refine business processes so that the applications supporting them can be seamlessly integrated across enterprises. The external business processes that control interactions among independent financial institutions are called *public* or *open* STP processes.

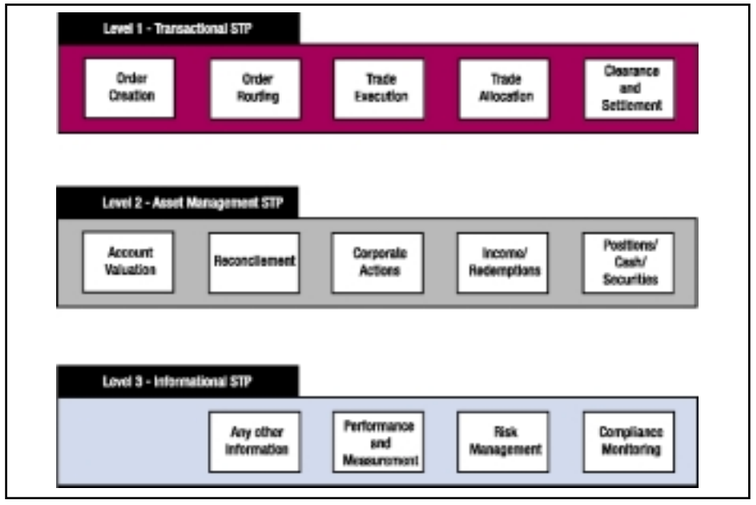


FIGURE 1 | **Different levels of STP**

What Is Orchestration?

Orchestration can be defined as the fabric used for the automation of business processes and long-running transactions comprised of loosely coupled services. Through orchestration, you can automate the following parameters of a business process:

- ***Why:*** The goal of a business process
- ***What:*** The activities/tasks that it involves, along with their input and output
- ***Who:*** Who performs the activities and where do the results go
- ***Where:*** The location – application, system, user – of the activities
- ***When:*** The order and timing of the activities

In the context of STP, orchestration would imply control of information flow for both private/closed as well as public/open processes (as defined earlier). With effective orchestration, financial institutions can become part of a unified business process flow. Unified business process flows would allow the dynamic sharing of trade-related information among internal applications of a company and external applications of trading partners through which all communication can be tracked and recorded. Since STP transactions are long running (e.g., can span multiple days), orchestration of unified business process flows becomes critical in ensuring the integrity and completion of automated business transactions.

The key orchestration requirements for STP will include identity management, stateful asynchronous interactions, flow coordination, business transaction management, and activity monitoring. Applying orchestration effectively to STP requires a dramatic reduction in the complexity associated with the above requirements needed for deploying and managing distributed business processes. This reduction in complexity will be achieved through the use of

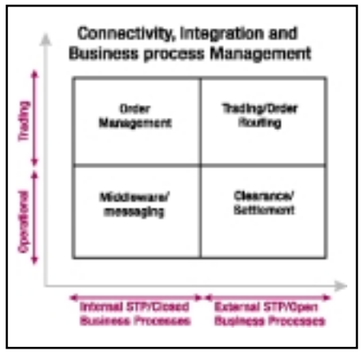


FIGURE 2 | **STP encompasses EAI, B2Bi, and BPM**

an orchestration server, which will be essential for addressing the critical requirements of orchestration for STP explained later.

Web Services as the Enabling Technology for Orchestration

Web services are sets of business operations that companies can make available through the intranet or Internet using open standards, such as SOAP, WSDL, and UDDI. It's worth mentioning that a Web service can be implemented as its own business process, which can then be executed under the control of an orchestration server. In that case, the process model is published as a Web service. Alternatively, a business process can be orchestrated through any number of Web services, which themselves can aggregate other Web services to provide a higher-level set of features. Web services, as an enabling technology, empowers orchestration to become the foundation of process-centric applications, implementing collaborative business processes and long-running transactions.

The elements that make Web services an ideal technology for orchestration are:

- Web services provide a clear separation of business process logic and the participating business services, thereby making the development, execution, and management of these services much easier.
- Web services fully leverage open standards, including HTTP; XML; Simple Object Access Protocol (SOAP); Web Services Description Language (WSDL); and Universal Discovery, Description, and Integration (UDDI).
- A typical business process may be supported by multiple diverse applications. It is virtually impossible to manage the information flow and execute the different tasks associated with it, which may require using APIs of other systems or exchanging messages with them, unless the underlying technology provides easy integration facilities. XML-based Web services are ideal technology for orchestration as they allow applications to communicate across the intranet or Internet in a platform- and language-independent fashion.

The key technologies and specifications that enable the orchestration of business processes as Web services include Business Process Execution Language for Web Services (BPEL4WS), Web Service Interactions (WS-Coordination, WS-Transaction), Business Transaction Protocol (BTP), Electronic Business XML (ebXML), and ISO15022 XML. These specifications are in a state of convergence from earlier efforts by individual companies, such as WSFL from IBM and XLANG from Microsoft.

Building an STP Solution over a Service-Oriented Framework

An STP solution has to connect multiple networks, applications, and platforms; has to be stable; and has to provide fast and accurate processing of complex transactions. The technology infrastructure should be extensible for today's and tomorrow's needs. Using this infrastructure, it should be relatively easy and cheap to use the opportunities of tomorrow to mass customize both financial and nonfinancial products and services. Further, it should assist the financial industry in migrating from IP-based environments to leveraging the full potential of the Internet. Finally, and most importantly, it should provide a solution that assists in the orchestration of business processes.

An SOA-based framework can enable financial companies to achieve their business goals by providing a service-based platform to integrate new and existing applications and systems with STP functionality, implementing industry standards, and building an infrastructure that would support the dynamic financial messaging required for continuous processing for all types of financial instruments. Service-oriented architecture can provide the foundation and Web services can provide the building blocks for application architecture in order to achieve seamless trade processing.

An SOA-based framework can provide support for multiple XML standards, such as ISO15022 and FpML, at the same time, and provide additional standards support without significant redevelopment effort. Using Web services as an enabling technology, STP-related problems and issues will shift from connectivity among different applications in-house and with trading partner applications to the content and structure of the information that is exchanged. The analogy here will be that Web services will define the standard postal mechanism along with the envelope and addressing format for exchanging letters. What is inside the envelope (the content of the letter) will be defined by the XML-based business process standard, such as ISO 15022 XML.

Orchestration in the Context of STP

Making Web services work for STP will be a two-step process. First, financial companies will have to publish STP-related internal and external Web services and then orchestrate them. Publishing would require taking a piece of business-related functionality that already exists within a back office, middle office, or front office application and making it available over the network using Web services standards so that it can be easily integrated into other applications. Orchestration would require designing, building, and executing an STP-related business process comprising of one or multiple Web services into a long-lived, multistep business transaction.

STP-related business processes could span multiple applications and levels (see Figure 1). Each of these systems can be written in a different language and run on a different platform. It is not possible and rational to rewrite these systems just because some of their functionality is part of an STP-related business process. Such functionalities will just have to be exposed and published as Web services standards–compliant interfaces, which can

then be orchestrated as part of a process-centric application.

Critical Requirements

The critical requirements for the orchestration of internal as well as external STP are:

- ***Real-Time Information***

In order to achieve STP, the trade information has to be passed between the buying entity, selling entity, exchanges, depository participants, and any other entity involved in the trade processing on a real-time basis at fast speeds. This requires either synchronous or asynchronous integration between applications running on different platforms (i.e., heterogeneous development and deployment environments) and possibly across the firewall.

- ***Security***

Secured interoperability holds the key for orchestrated STP to become a successful initiative. The security requirements for internal STP are almost a subset of those of external STP. External STP involves significant security risks as it involves the use of the Internet or VPNs, which mandates two levels of security. First, external STP necessitates opening up corporate firewalls to enable cross boundary communication between enterprises. Thus, financial companies have to secure their internal network against malicious attacks through these open ports. Second, the data transmitted over the Internet or any other mode has to be secured. The data may contain classified information, such as corporate information, trade information, and settlement information, and thus cannot be left unguarded.

- ***Information Flow Modeling***

In order for orchestration for STP to work, it is imperative that you be able to model and create the orchestration logic for the business processes. Process modeling involves drawing a flow diagram that links resources, logic, and movement of information between systems. Subprocesses are used when the main process becomes too complex and needs to be broken down to keep it simple. An orchestrated environment should provide the capability of binding the components (process models; real entities such as companies, organizations, or people; source and target systems of the trading partners) of a business process together.

- ***Information Flow Execution***

Any orchestration for STP will not be possible without a robust, scalable, and high-performance orchestration server. The orchestration server uses the process models to execute the processes, directing the flow of information as needed to and from a financial company's internal IT systems (front, middle, and back office systems) and over the Internet to the IT systems of its trading partners, including custodians, exchanges, clearing houses, and banks.

- ***Activity Monitoring***

Monitoring and reporting of all activities involved in an orchestrated end-to-end business process is essential for STP. This is important for several reasons, including audit requirements, regulatory requirements, performance management, and error resolution of underlying Web services.

- ***Exceptions and Transaction Management***

This is probably one of the most important requirements for achieving an orchestrated STP paradigm. It defines and scopes the capability to process automated transactions while handling exceptions either programmatically or manually. It is worth mentioning that STP will not be able to completely eliminate human intervention for all business processes, especially in cases of exception handling. There should be hooks provided in the orchestrated STP-related business processes to allow for human intervention for providing information or making decisions wherever necessary, as defined by the orchestration logic.

- ***Transparency***

Business process transparency is an extremely important requirement of STP. At any stage of the trade cycle and at any time, any entity (if entitled) should be able to determine the state of the transaction.

- ***Standardization of Business Processes***

It's only through the standardization of business processes that key requirements of STP – automation and common processing platforms – could be achieved. Without standardization, companies will not be able to leverage reusable solutions across business applications for either EAI or B2Bi. Business processes orchestrated through a Web services foundation need to be interoperable to ensure that new interactions and changes to the business process can be deployed quickly.

Let Your Imagination Run Wild!

Imagine that it's possible to customize a range of financial and nonfinancial product packages suited for each customer's specifications dynamically in real time. Imagine that it's possible to target products and bundle them together, based on end users' interests in real time. Imagine that it's possible to connect to any clearing corporation without new technical and business requirements. Imagine that you can use the back-office services of any company without the overhead of several days worth of setup. Imagine that it's possible to electronically create new financial instruments through collaboration across different companies. Imagine that it's possible to automate all activities related to corporate actions. Imagine that it's possible to forget about making changes for T + 1 and target T – real time, now.

Can these dreams ever become a reality? Yes, it's possible. Not immediately, though. It certainly can happen and in all likelihood it will be SOA-based frameworks that will make it happen through the orchestration of business processes. This is directly dependent on three technical factors: how soon can Web services standards mature; how soon can the XML standards representing business processes, such as ISO 15022 XML, for the financial vertical industry mature; and how soon can the tools and servers that will enable the orchestration of Web services mature?

We're optimistic that this will come true sooner rather than later. It's only by imagining the evolving technology that we can set the benchmarks and direction for its growth, future research, and last but not least, adoption.

Reviewed by Joseph A. Mitchko

webMethods Version 4.6

Web services-based integration platform

About the Author:
Joe Mitchko is the product review editor for Web Services Journal. JMITCHKO@RCN.COM

webMethods

COMPANY INFO
webMethods, Inc.
3930 Pender Drive
Fairfax, VA 22030
USA
Tel: 703 460-2500
Fax: 703 460-2599
Web: www.webmethods.com
E-mail: salesinquiry@webmethods.com

LICENSING INFORMATION
Platform licensing is CPU-based.
A 60-day evaluation copy is available at:
http://evals.webmethods.com

TESTING ENVIRONMENT
OS: Windows-2000 SP3
Hardware: Athlon 1GHZ / 768M RAM

Information technology sometimes reminds me of the magic industry. The first time a new and exciting act is performed, it creates a level of stir in the industry, leading to the eagerness of other magicians to perform it in their acts. Over time, the secret is gradually shared (or obtained by illicit means) among fellow magicians until eventually everyone is performing it. You can carry this analogy to Web services. For what started out as a trickle, we now have a myriad of products implementing SOAP-related protocols – old hat at this point.

But the core issue regarding Web services has nothing to do with adding a SOAP and WSDL interface to a system. It's something more mundane and time intensive, and has been around since the early days of computing: integration, getting all of the various external components to act as a single system. Most of the time, integration isn't a simple task, and with Web services, it's made more complex by the increased number of business partners involved and the need for complex, long-lived service transactions. Add to the equation the need for highly robust and redundant platforms for these services and the picture becomes more complex. With all this complexity, products such as webMethods are vital for Web service integration.

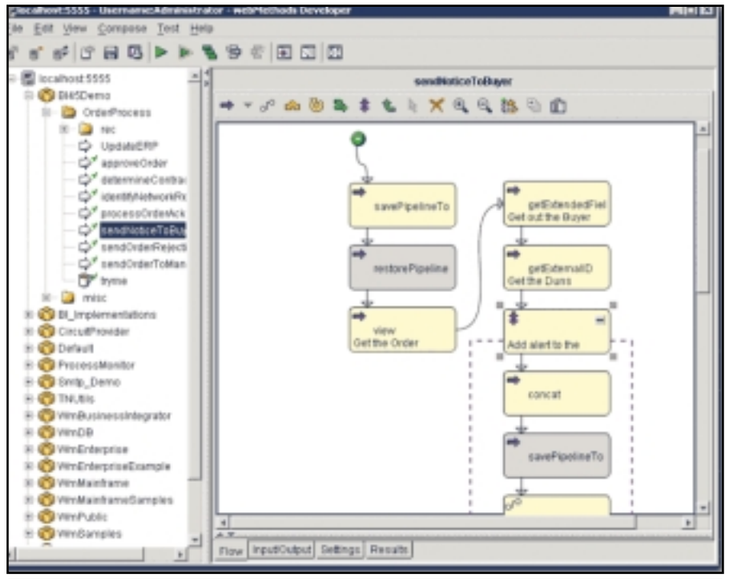


FIGURE 1 | **The flow of data through various services**

Overview

The webMethods 4.6 platform is a comprehensive business integration solution that enables companies to manage and automate business processes across applications, trading partner interfaces, and human workflows. With its service-oriented architecture, webMethods allows Web services to be incorporated seamlessly into these business process flows. The webMethods Integration Server is a service-oriented processing engine capable of providing communications support; integration logic; and enterprise-class scalability, reliability, security, and transactional integrity for enterprise Web services. The webMethods Developer is an integrated Web services development and deployment environment, providing a point-and-click interface for creating, integrating, testing, and deploying Web services. webMethods also provides a set of optional adapters for interfacing with external resources.

Creating a Web Service

What's interesting about webMethods is how you go about creating a Web service. Since webMethods has a service-based architecture (it has since its inception), and the integration process is comprised of creating various flow services, you don't need to follow a separate process to create and deploy a Web service. It's automatically generated for you. All services managed by the Integration Server are therefore available as Web services.

To assist you in testing and publishing a Web service, webMethods provides a Web service directory consisting of a testing facility and a WSDL viewer.

The testing facility generates an input form for the service call, calls the Web service, and displays the Web service response. You can also look at the actual request and response XML documents and view the WSDL. Both RPC and message-based SOAP service requests are supported by the webMethods Integration Engine. webMethods is capable of handling SOAP request and response messages over the HTTP and HTTPS network protocols.

Let It Flow

The flow model or language is the heart of a webMethods-implemented Web service, and provides the developer with the means to assemble various services, message channels, and database transactions into a new service. Services are built upon other services until you've completed the top-level service. For Web service development, this means that you can piece together various external resources along with business flow logic to produce complex Web services.

The integration work is performed in the webMethods Developer module and is primarily a model-driven process.

The flow model includes built-in conditional flow steps that you can use to apply business logic, thereby directing the flow of information within a flow service. Examples of the flow steps include branch, loop, repeat, and exit.

You also have the ability to add pre-existing external Web services to the flow diagram, thereby allowing a Web service to utilize external Web services (monetary exchange lookups, for example). webMethods provides a rich set of built-in services that handle a variety of internal or external processes. Some examples of built-in services include LDAP support, database access, and MIME header processing.

Integration Engine

The Integration Engine (IE) manages all services in the webMethods platform and processes the various flows that are resident in an active service. It's essentially where the services that are created by the developer are deployed. Transaction management is handled by the Job Manager in the IE and is directly associated with guaranteed message delivery. In addition, various time-outs can be set within a service to roll back transactions that have expired.

The IE is capable of operating in a clustered environment and supports load balancing and other high-availability features. Last, a browser-based administrative console provides configuration, reporting, and auditing, as well as other administrative features.

Typical Application

The webMethods Integration Server is typically situated behind the corporate firewall, integrating various heterogeneous systems into internal and public-facing Web services. A typical deployment may involve integrating customer information found in an ERP system with a J2EE-based intranet application into a set of new Web services. In this case, the Integration Server will integrate with systems located behind the firewall and optionally publish "secure" Web services over the Internet for public consumption. I quote the word secure because there is currently no standard regarding secure Web services. Another application would be to use the Integration Server to buffer existing systems in the enterprise and present them to an application server as standardized SOAP messages. In this setup, the application server would only have to deal with one interface, SOAP, for all of the back-end service calls.

Adapters

webMethods integrates ERP/CRM packages, mainframe applications, EDI/e-standards, etc., providing a variety of adapters, including those for packaged software from SAP, J.D. Edwards, Siebel, EDI, Baan, PeopleSoft, Oracle Apps, and BroadVision; databases such as JDBC and ODBC; messaging middleware, including MQSeries, MSMQ, and JMS; and Java application servers such as J2EE, EJB, and JCA. The product also offers an adapter development kit for developing custom adapters.

First Impressions

Installing webMethods was straightforward. I suggest you read through some of the documentation first and go through some of the tutorials to get acquainted with the service-oriented process and the flow modeling process. Again, to develop a Web service, all you need to do is develop a flow service; the rest is automatic. The Web service directory facility was easy to use, and I was able to test several Web services using the facility without any difficulty. A lot of additional functionality is outlined in the PDF documentation, so look through the various developer guides to get an idea of some of the additional Web service capabilities available, including custom SOAP handling, security and access control, and so on. I found that there's a bit of a learning curve involved in getting up to speed with the flow language, but anyone with programming experience should be able to pick it up easily. There are plenty of demonstration services and tutorials to help you along.

You can download a 60-day, fully functioning trial version from the webMethods site. You'll need to register prior to downloading.

Summary

webMethods was in the service-based integration business long before Web services arrived on the scene and, in fact, was involved early on in the standards process. The product enjoys a wide customer base and is one of the leading e-commerce platforms. The support for Web services in the 4.6 release is a natural fit for a service-based integration platform and introduces a new level of integration capabilities for deploying public, long-transaction, message-based Web services. It's this type of capability that will propel Web services into its rightful place in information technology history.

WEB SERVICES SECURITY

The key is unification

Written by Lakshmi Hanspal

Web services is a promising technology for achieving B2B enterprise application integration. However, the challenge of ensuring security in a Web services environment stands as a major obstacle to general acceptance of the technology.

In the first part of this series (***WSJ***, Vol. 2, issue 10), I discussed traditional approaches to securing Web services and the shortfalls of these approaches, demonstrating the need for a comprehensive, standards-based security framework with a purpose-built solution, such as SOAP Content Inspection, to completely secure Web services.

In this article, I further explore the world of federation in B2B Web services. Distributed-component computing allows the sharing of information among enterprises. But enterprise security policies are likely to be different (say, between a hospital and a bank), which means that data sharing requires translations between enterprise policies. We will also discuss how a security framework approach using the Web Services Proxy unifies federated security

AUTHOR BIO:

Lakshmi Hanspal is a security architect with Quadrasis, a business unit of Hitachi Computer Products (America). She has several years of extensive experience in the architecture, design, and development of security and directory services, including a patent on directory service schema. Lakshmi also contributes to technical committees at JCP and OASIS and is a Sun Certified Java programmer and J2EE Architect. SOLUTIONS@QUADRASIS.COM

Web Services and Federation

The advent of Web services has altered the economics of application integration by expanding the possibility of EAI (enterprise application integration) to not only high-dollar, large trading partner enterprises, but also to small and medium-size companies that typically lack the sizable IT budgets of their much larger counterparts. By standardizing all APIs as XML format, Web services provides a starting point to enable cooperative computing and federation of applications.

So What's The Catch?

Such federation opens a Pandora's box of security issues. Web services have a huge problem in that they are too open. Companies will need to limit access to their valuable resources, such as patient records, credit card numbers, manufacturing designs, etc. Hence, there is a need to collaborate and share information, but not at the expense of giving away corporate assets.

What Is Federated Security?

The Internet is a prime vehicle for business, community, and personal interactions. The notion of security is a crucial component of this vehicle. Security on the Internet is fragmented across various business providers, such as differing and disparate user identities, security mechanisms, and business access policies. This fragmentation yields isolated services and/or poorly connected business-to-customer relationships and experiences.

Federated security is the key to reducing this fragmentation and realizing new business taxonomies and strategic opportunities, coupled with new economies of scale. A good example is the notion of several different companies cooperating on a single transaction (such as making a reservation that involves airlines, hotel rooms and rental cars: the federated identity could mean a single sign-on and linking all these transactions on different sites). So at its heart, federation means trusting sources to come across your borders and opening up your services to them. Federated identities are portable across normally impervious enterprise boundaries. Federated single-sign on allows users to sign on once with a member of an affiliated group of organizations, and subsequently use various services offered by the group without signing on again

Business Trust Models

Federation works on the principle of a business trust model. As more and more businesses execute their transactions and move XML data across the Internet, they face increasing security challenges as they attempt to protect business-critical data and access to their business with increasingly complex trading relationships. Web services that span multiple trading partners usually require interaction between many entities (users, application components) and integration across different corporate security policies for authentication and authorization.

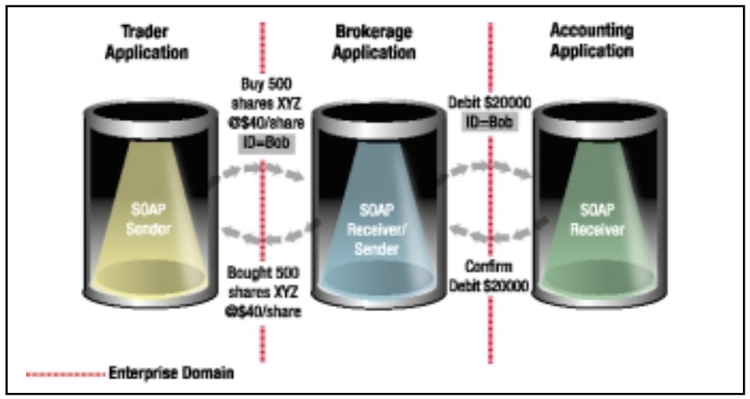


FIGURE 1 | **Trusted business computing**

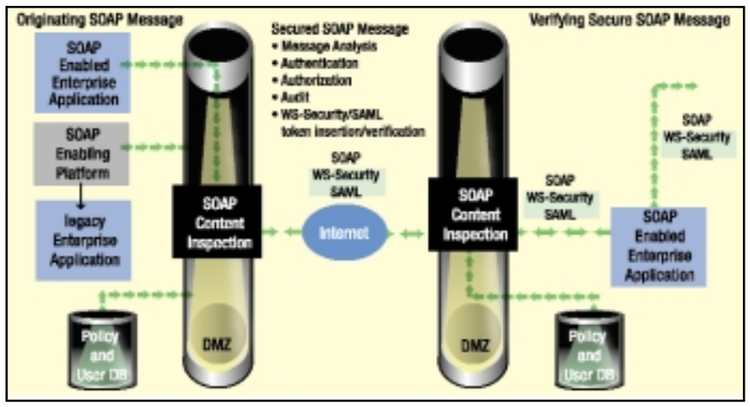


FIGURE 2 | **Integrating Trust Access Enterprises – WS Proxy**

Consider a trading transaction that spans multiple business processes across multiple enterprise domains (see Figure 1).

A trade request to buy 500 shares of Company XYZ is sent from a trader application in one enterprise domain to a brokerage application at a brokerage company. On approval, the trade is executed and the settlement request is sent to the accounting application (in another company) to debit the appropriate amount from the customer's account. Confirmation of the debit and execution of trade is then sent back as a response to the trader application.

Web services expose the internal network to requests via HTTP that passes through firewalls, requiring trustworthy connections of multitier applications – applications that proxy (delegate) the request to others downstream. Although single sign-on facilitates the ease of use of such application chaining, it alone does not secure application chains; typically it only applies within a corporate or business unit environment. Furthermore, different companies use different security products that don't interoperate. Consequently, the lack of consistent user identities, authentication and authorization policies, and implementations introduce a level of complexity that must be addressed in order to define business trust levels in a federated environment.

First- and Last-Mile Problems

Web services create a jungle of tangled integration processes, from the integration of disparate business applications to the integration of back-end business systems.

Access to most business Web services will need to be restricted to only authenticated clients. Currently, Web services standards efforts do not prescribe any standard mechanisms or measures to describe or package authentication evidence such as userID/ password, certificate, biometric information, etc.) to present to authentication services to authenticate the entity (user, application component, etc.). This leads to lack of bridging for the "first mile" of Web services access.

Second, emerging Web services standards tend to ignore the fact that there are many, many applications and programs in the world that are not XML-centric. Back-end systems are full of applications and programs with proprietary data and interfaces predating the XML era. Web services standards currently do not address integration in this area. This leads to lack of integration with the "last mile," and points to the need for a secure integration bridge between the land of legacy and the promised land of Web services.

Enterprise Policy Mapping

Web services security standards address standardization of data schema and formats. Given that these standards are approved and adhered to, there still remains the issue of data mapping. Although there are emerging standards such as XACML (Extensible Access Control Markup Language) to define standard access-control policies, the semantics of data in these policies is left to interpretation. A simple example is illustrated by the notion of enterprise user roles: the roles of users in each enterprise could be semantically different (each role, e.g., buyer, purchaser, customer, could all mean the same thing in different companies, and yet have different rights in each domain). Enterprise security policies could use enterprise-specific roles for their access-control policy rules. Even within an enterprise, different business units could use different application-specific roles. Here there arises a need for role mapping that can determine equivalency or a unique translation of roles.

Unified Security via WS Proxy

To make Web services successful, security must be in place to handle federation. Companies will not open up corporate networks without proper security. There is a need to protect against external (such as from the Internet) and internal attacks (such as fraudulent trades and trapdoors left by ex-employees to access corporate systems). Furthermore, there is a need for an architectural approach to unify security policy and implementation across different business units and companies.

Enterprise Application Security Integration (EASI) is a standards-based, vendor-neutral security framework that unifies the patchwork of security products and services deployed within the enterprise.

Current and emerging XML Web services security standards include:

- WS-Security for SOAP message security
- SAML (Security Assertion Markup Language) for exchange of user credentials between components
- The Liberty Alliance Project for federated network identity and single sign-on and sign-out.

So if all security products will eventually use these standards, why do we need EASI?

- ***Existing standards do not cover all tiers of computing:*** Many of the standards play in the middle tier. Hence there are always legacy security needs that can be answered through EASI.
- ***Standards do not guarantee compatibility:*** The hype of standards associated with SOAP and Web services comes with the associated problem of the splintering of those standards. Pseudo-compliance or noncompliance creates pseudo or hybrid systems. EASI helps to connect countless data formats, standards and pseudo-standards.
- ***Standards are moving targets:*** EASI allows applications to take advantage of evolving security services while the framework deals with such complex moving targets because of its plug-and-play approach.
- ***Federation means enterprises also need to translate and map security data:*** Such mapping services are provided by EASI, which insulates the user from the complexities involved and eliminates disconnects in the existing patchwork of point solutions.

In B2B scenarios, Web services messages may travel over any number of connections and potentially traverse many intermediaries before reaching their destination. In order to support this decoupled interaction, connection-oriented security alone is insufficient, requiring the addition of security on a per-message basis. Consequently, messages must be self-contained with respect to security information and carry all the necessary information and credentials with them. Inspection of message schema is also required to prevent unpredictable behavior of Web services due to poorly formed messages.

Federation also frustrates accountability (audit). It is harder to keep track of what happened and who penetrated your defenses when an attack could come from outside your sphere of control and from systems that implement different security audit policies and track different user and application identities. Accountability in a federated environment requires a framework to unify audit policies. Audit records should be implemented to unify the meaning of audit events across different auditing services and to allow a means to correlate information collected in an audit repository with information collected in another to understand what actually happened across a distributed transaction in a federated environment.

Integrating Trust Across Enterprises: The Web Service Proxy

A key service of the EASI Framework is the Web Service Proxy (WS Proxy), a flexible, standards-based solution that secures SOAP-based transactions for a range of enterprise B2B applications (see Figure 2).

The WS Proxy uses services provided by the underlying EASI Framework. Its combination of services supports the three principles of Web services security:

- ***Trust no one:*** Requests traverse many domains and applications. Hence a single compromised point may make the system vulnerable if additional checks are not in place. The WS Proxy supports client authentication, thereby providing a bridging solution to the "first mile" problem.
- ***Enable interoperability:*** A single vendor solution does not solve all aspects of security; customers and partners will pick different products, requiring interoperability with other Web services and applications using incompatible security The WS Proxy (through the EASI Framework) provides connectivity to back-end systems for request processing, thus eliminating the bottleneck that these systems pose for Web Services and bridging the "last mile."
- ***Modularize security:*** Web services security technology will continue to evolve, so there is a need to maintain flexibility to mix and match emerging security technologies without requiring recoding of applications.

TABLE 1: The WS Proxy in various domains

	Use Web services	WS Proxy
Intranet	Internally for: • Cross department access to services • Application Integration	• Secures access to Web services resources from other departments • Secure interapplication communication
Federated Extranet	To integrate applications and services with: • Trading partners • Branch offices	Federated Trust eliminates duplication of user and policy information
Internet	To deploy new: • Application services accessible to broad audience • UDDI registered services	• Allows broad access to services • Provides authentication and authorization services

The WS Proxy in Various Domains

The WS Proxy provides security for Web services deployed in different domains (see Table 1).

In each domain, the domain security policies are applied for authentication, authorization, audit , etc. Domain policy metadata relationships can be established using WS Proxy administration. These relationships can be used to translate between differing domain policy data such as roles.

Conclusion

Web services demand attention to federation of services and security. Developing federated security policy support for Web services is an evolving part of EASI.

Comprehensive SOAP/XML message security is more than encryption. It requires an application-level security gateway as a flexible, transparent solution that performs message analysis as well as authentication, authorization, and auditing services to protect business-critical Web services.

The EASI Framework, along with the WS Proxy, enables organizations to leverage their existing investments and easily develop new business relationships with suppliers, vendors and customers, while ensuring adherence to corporate policies to achieve unified security in federated environments. WS Proxy provides comprehensive inspection of messages (message validation, message integrity, and message-origin authentication). Critical integration with enterprise security services (authentication, authorization, audit, SAML interoperability) and SOAP security ensure end-to-end protection and assurance.

Written by Karen Thure and Frank Lacombe

The Sun ONE Architecture: Why Care?

All of the standards, technologies, and products needed to support the pioneering Web services that are being built today

The Sun ONE architecture is Sun's software vision, architecture, platform, and expertise for solving many of today's enterprise integration, interoperability, and development issues. Based on the Java 2 Platform, Enterprise Edition (J2EE), it provides an easy evolution from nonintegrated enterprise applications to fully integrated and interoperable Web services.

A software developer will find it useful to learn the Sun ONE architecture and discover how it can solve integration and interoperability problems. Its associated technologies and products provide a wide range of automated services that significantly lower the costs of building Web applications and clients.

This article provides an overview of the architecture. For an in-depth analysis, see the Sun ONE Architecture Guide, which contains discussions of the architecture's standards and protocols, along with descriptions of the technologies and products that implement them. For practical, how-to information we provide a list of more specialized articles at the end of this piece.

Whatever your place in the industry – CTO, industry analyst, IT manager, developer, or other software professional – this overview will give you a good grasp of the Sun ONE architecture and an understanding of its value to software developers.

Services on Demand

Sun coined the phrase "Services on Demand" to signify anytime computing anywhere, to anyone, using any device. The Services on Demand vision is of a comprehensive framework encompassing traditional Internet-based services, such as security, authentication, and directory, along with more advanced capabilities, including virtualized storage and composite services (those created by combining separate services). This vision encompasses these technologies:

- Current Web applications, Web services, and Java Web client applications
- Upcoming "smart" (as in contextually enhanced) and federated services such as those that recognize human user, resource, and application contexts
- Potential distributed computing technologies such as fully integrated e-commerce

Creating applications and services for the huge number of always-connected users of these technologies, and developing the infrastructure to support them, will offer rewards to those who are prepared to meet the challenges. The good news is that your existing understanding of Java technology, object-oriented design, and related types of programming expertise are the foundation for your growing skills. Essentially, you're already more than halfway up the path that eventually will lead to the world of fully integrated e-commerce.

A multitiered platform consisting of Sun ONE middleware products, the Solaris Operating Environment, and J2EE technology, the Sun ONE architecture allows you to leverage your legacy software and current programming skills.

Web Services: Evolution, Not Revolution

Within the realm of Services on Demand, "Web services" are a subset, but one that has recently gotten a lot of attention in the software industry. Today, many people are skeptical of the over-hyped Web services "revolution." Such skepticism is warranted to a degree, because the prerequisite industry-wide standards; Web infrastructure; and key technologies, such as network identity and single sign-on, are still immature.

"The evolutionary nature of Web-based software was a major consideration in the development of the Sun ONE stack," says Hal Jespersen, CTO of Sun ONE Products and chair of the Sun ONE Architecture Council. "Instead of asking developers to adopt a new application paradigm such as Microsoft's .NET, Sun ONE enables them to build Web services using the same familiar Java technologies that they're currently using to develop Web applications. These include servlets, Enterprise JavaBeans technology, and all of the other open standards associated with the J2EE programming model."

Sun ONE is Sun's architecture for ultimately delivering Services on Demand.

The Sun ONE Architecture: A Quick Look at the Stack

The Sun ONE architecture consists of an integrated stack of standards and technologies in three layers (see Figure 1). In the top layer are the elements used to create, assemble, deploy, and test Services on Demand. At the bottom are identity, security, policy, and management, along with hardware/software platform support. The middle of the stack is the center of the Sun ONE architecture: Service Delivery (presentation logic), Service Container (business logic), and Service Integration (back-end data-access logic).

Author Bios

Karen Thure is director of Creative Services for KnowledgeTree Systems, Inc., a technical communications outsourcing firm serving the high-tech community since 1980. The author of two books and 19 national magazine articles, she has been a technical writer and manager for more than 16 years.

Frank Lacombe is founder and president of KnowledgeTree Systems, Inc., a technical communications outsourcing firm serving the high tech community since 1980. Active for many years in alternative education, he serves as a trustee of the Santa Cruz Waldorf School, of which he is also the president.

Sun ONE Standards and Products

The Sun ONE architecture is based on a number of standards for APIs and protocols, including Java technology, XML, and Web standards (see Figure 2). Figure 3 shows how Sun ONE products fit into the architecture. Because the Sun ONE platform is integratable, products from other vendors that conform to open standards can be used interchangeably. On the other hand, the integrated nature of the all-Sun ONE product stack is an important value proposition for many developers and enterprise IT departments. Sun provides the Sun ONE integrated products as part of the Sun ONE Developer Platform.

Tools of the Trade: The Service-Building Layer

At the top of the Sun ONE stack, the Sun ONE Studio IDE (formerly Forte for Java) provides a full suite of tools designed to support the Sun ONE architecture. Based on the open-source NetBeans platform, the Sun ONE Studio product delivers the modules, wizards, templates and generators that enable development, assembly, and deployment of J2EE components and standards-based Web service applications to the Sun ONE Application Server in a team-oriented environment. It is ideal for building and deploying Web services across multiple hardware and software platforms. This modular, extensible IDE enables fast adoption of new technologies from Sun, Sun's partners, and the community.

Developers can customize Sun ONE Studio developer tools by:

- Adding modules that come with the tools to provide alternative editors, debuggers, or other functions
- Writing their own modules to add new features or replace functionality
- Enabling or disabling various modules

The Sun ONE Studio IDE is included with the Sun ONE Developer Platform Technology Preview. This development, test, and deployment environment is especially suited to enterprise projects such as portals that provide many types of customized user services. The Sun ONE Developer Platform Technology Preview includes evaluation copies of complete application, security, and enterprise information systems (EIS) integration functions, along with portal-building capacity. It also includes an evaluation copy of the Sun ONE Application Framework.

In addition, the Sun ONE Application Framework, a best-practices implementation of the Model/View/Controller (MVC) design model, provides developers with a proven pattern to employ in the creation of Web applications and Web services. Using the Sun ONE Studio developer tools in conjunction with the Sun ONE Application Framework allows Java component developers and form-based application builders to work together to deliver their products.

Where the Action Is: Delivery, Runtime, and Integration

The middle layer of the Sun ONE stack consists of three important areas: Service Delivery, the Service Container, and Service Integration (see Figure 4).

Service Delivery: It's All in the Presentation

The Service Delivery portion of the Sun ONE stack contains the components that developers use to communicate with users. Within the Sun ONE architecture, Service Delivery is implemented by the Sun ONE Application Server and the Sun ONE Portal Server.

The Sun ONE Application Server provides a platform and infrastructure for developing and deploying Web applications. By serving as a single integration point between many applications, it greatly reduces integration complexity. In addition to free-

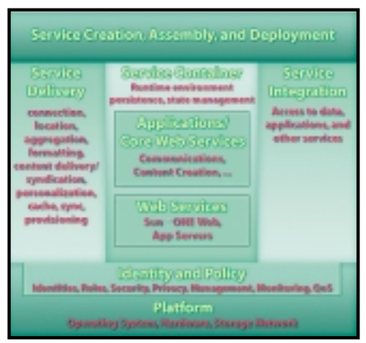


FIGURE 1 **The Sun ONE Platform for creating Services on Demand**

FIGURE 2 **Standards associated with the Sun ONE Architecture**

FIGURE 3 **Products associated with the Sun ONE Architecture**

dom from system-level programming, it provides enterprise integration and business process automation services.

The Sun ONE Portal Server is an identity-enabled platform and infrastructure for developing and deploying many types of secure portals. It offers a wide variety of portal-building services, including user and community management, personalization, aggregation, security, integration, and search.

J2EE technology provides a presentation framework as defined by JavaServer Pages and Java servlet technologies. This framework acts as a foundation for gathering information from both end users and the business layer of an application. It also generates the user interface and processes all user interactions with it. If you prefer a prebuilt presentation framework, you can use the Sun ONE Application Framework, which is integrated with the Sun ONE Application Server.

Another set of Service Delivery elements helps Java Web client developers deliver applications to a variety of client devices, including desktop computers, mobile phones, PDAs, home gateway computers, and set-top boxes. The Mobile Access Pack of the Sun ONE Portal Server provides services for securely connecting various mobile devices and users to their portal.

Java Web client technology is spurring the development of the JSR-124 standard (also known as the Java Vending Machine) that allows users to purchase or rent applications appropriate to the device they're using, their location, and their personal preferences.

FIGURE 4 | **Sun ONE Service Container**

The Service Container: Runtime and More

The Service Container is the implementation of J2EE technology running in an application server that supports the Java XML APIs. The Sun ONE Application Server is Sun's implementation of the Service Container. It manages software – including applications and services – across a broad spectrum of Web and enterprise software requirements. Along with the runtime environment, the Service Container includes persistence and state management services.

As a product that delivers the Service Container, the Sun ONE Application Server provides mechanisms for transaction management and connection management, including such functions as connection pooling and security. By serving as a single integration point between many applications, it greatly reduces integration complexity.

The Sun ONE Application Framework runs in the Application Server. A best practices implementation of the familiar MVC design model, it can be incorporated into applications developed on the J2EE platform.

In addition to the core Java 2 platform features, the J2EE technology includes support for Enterprise JavaBeans components and Java servlet, JavaServer Pages, and XML technologies. The Java BluePrints Pet Store sample application illustrates how Web services can be added to an existing application using the J2EE platform.

Sun is currently working on a set of "prebuilt" core Web services. These will provide a set of system- and application-level functions that supply e-commerce programs with contextual information, such as the user's location and access device. Other core Web services will expose the main functionality of communications applications, such as e-mail, calendar, address book, and conferencing.

Service Integration: Leveraging Your Legacy Systems

Web services facilitates integration and interoperability. Using this model of software development, and by building integration adapters and connectors, you can leverage your legacy systems, third-party products such as workflow packages, and even other enterprise technologies such as Microsoft .NET.

The Service Integration features use many technologies, including J2EE Connector Architecture technology and the Java Database Connectivity (JDBC) API, to integrate legacy applications and databases, access service registries, and perform electronic commerce integration within and between enterprises.

The Sun ONE Connector Builder is a software tool for building Web services–ready connectors to integrate J2EE and Web applications with existing enterprise systems and legacy applications. It accomplishes this by generating resource adapters that are compliant with the J2EE Connector Architecture and that can be deployed in the Sun ONE Application Server. These resource adapters can also be run in a Web container for loosely coupled access using the SOAP specification.

The Sun ONE Integration Server, EAI Edition, enables enterprises to integrate packaged, custom, legacy, and new applications based on Java technology into a workflow to complete business processes. It includes a Web services integration framework, which also works with the Portal Server. By automating such tasks as building and deploying XML adapters and by providing support for the Web Services Description Language (WSDL), this framework makes it easier to integrate back-end systems as Web services or Web client applications.

The Sun ONE Integration Server, B2B Edition, enables enterprises to integrate e-commerce applications for partners and customers, whether they are accessed over the Internet or on private networks. In addition to allowing the encrypted transmission of documents and messages among heterogeneous trading partners, this server can transform information from one data format into another – for example, from HTML to XML. Its integration broker also includes business process management and message transport.

This edition also includes a Process Designer tool that allows system integrators to build a workflow package. Local applications, Web applications, and Web services can use this package to operate together smoothly.

Web services are also well on their way to becoming a major integration strategy in themselves. The native support for Web services inherent in the Sun ONE architecture upholds their loosely coupled integration mechanism.

The maturation of the Electronic Business eXtensible Markup Language (ebXML) collection of Web services specifications will be critical to making Web services a common mode of integration. The Sun ONE architecture embraces both the Web services native specifications and the business extensions offered by ebXML. The Sun ONE Message Queue technology is based on the Java Message Service (JMS) specification. As the ebXML messaging specification matures, Sun plans to expand the Message Queue product to include interfaces to that framework.

Security for All: The Identity and Policy Layer

A major part of Sun ONE security consists of Identity and Policy Services, which are near the bottom layer of the stack. Interacting with most of the higher-level components, they include these broad categories of services:

- Identities, roles, and security for users, groups of users, and other system objects
- Federated identity systems such as the Liberty Alliance Project
- Management services, which include both systems and applications management

Within the Sun ONE architecture, the Sun ONE Identity Server provides an identity system that includes access management, identity administration, and directory services. The Sun ONE Directory Server serves as a central repository for storing and managing identity profiles, access privileges, and application and network resource information.

Built on top of the Directory Server, the UDDI-based Registry Server lets enterprises register Web services, thus allowing their services and business processes to be identified, described, and integrated on the Internet. Other Sun ONE platform services provide authentication, Web single sign-on, identity and policy management, logging, and audit.

The Sun ONE architecture currently supports industry public-key infrastructure (PKI) standards as well as the Kerberos authentication protocol and the UDDI specification. Sun recently announced its support of the Liberty Alliance version 1.0 specification for an open, federated identity solution.

Within the Identity and Policy layer, a set of management services provides both the framework and interfaces needed to manage Web services and system resources throughout the Sun ONE platform. These management services also allow your Web services to employ the same management methods and interfaces that are used in the stack.

A Solid Foundation: The Platform Layer

At the bottom of the stack is the Platform layer, which includes Solaris and other operating environments, such as Windows and Linux, along with hardware and storage. This is also the location of the networking platform.

Platform services allocate and manage the resources that host the higher-level service layers in the Sun ONE architecture. Their functions range from furnishing network connection interfaces to providing security facilities and printing services.

Getting a Jump-Start: Introductory Tools and Services

For an introduction to the Sun ONE architecture and the integrated suite of Sun products that implement the architecture, many developers are electing to order the Sun ONE Developer Platform, which contains evaluation copies of the tools and servers described here. It also contains a sample end-to-end application that illustrates how Sun ONE products can be used by trading partners to conduct e-commerce across the Internet. The Sun ONE Developer Platform is available by ordering the Sun ONE Starter Kit at wwws.sun.com/software/sunone/starterkit/index.html.

For a hands-on introduction to building Web services, many experienced Java technology developers are downloading the Java Web Services Developer Pack (Java WSDP). This contains the Java XML APIs along with a set of ready-to-use tools necessary for building, testing, and deploying Web applications, XML Web applications, and Web services on the Java platform. Information about the Java WSDP is located at http://java.sun.com/webservices/webservicespack.html.

The Future of the Sun ONE Architecture

The Sun ONE architecture currently includes all of the standards, technologies, and products necessary to support the pioneering Web services that are being prototyped, tested, and even deployed today. Through frequent updates and releases, the Sun ONE products will expand to accommodate increasingly sophisticated distributed computing models.

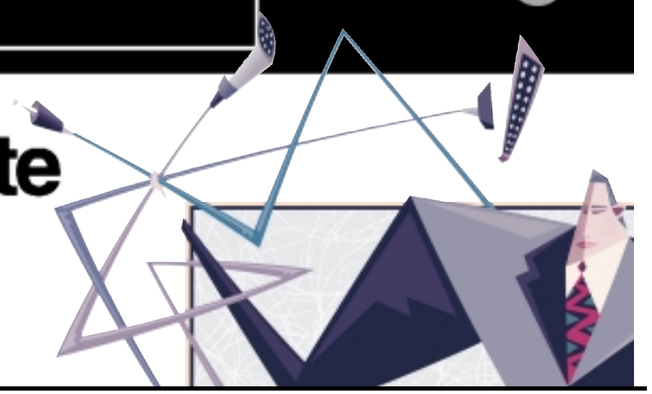
Related Resources

- Sun provides a developer Web site for the Sun ONE platform at http://developer.sun.com/sunone.
- *Sun ONE Architecture Guide:* wwws.sun.com/software/sunone/docs/arch/index.html
- *Sun ONE Developer Platform:* www.sun.com/developer
- *Java Web Services Developer Pack:* http://java.sun.com/webservices/webservicespack.html
- *Getting Started in Developing Web Services:* http://developer.sun.com/sunone/building/howto/gettingstarted.html
- *Web Services Distilled:* http://developer.sun.com/sunone/platform/technologies/wsdistilled.jsp

Written by Aravilli Srinivasa Rao

Develop a Private UDDI Registry

The Java API for XML Messaging



UDDI (Universal Description, Discovery, and Integration) is widely accepted as the standard for Web services publishing, querying, and discovery. A UDDI registry allows you to publish and browse Web service references via SOAP (Simple Object Access Protocol) and HTTP interfaces. The structure of the UDDI specification allows the possibility of private UDDI nodes or a private UDDI registry. A private UDDI node can implement all the UDDI APIs (Inquiry and Publish) as defined by the UDDI specification, but it doesn't participate in the replication scheme, nor does it reside within intranets, extranets, or private networks on the Internet. In this article, I'll show you how to develop a private UDDI registry using the Java API for XML Messaging (JAXM).

AUTHOR BIO:

Aravilli Srinivasa Rao, a software analyst at Hewlett-Packard, is technical lead for the development of HP's public UDDI registry. He has a master's degree in computer applications and six years of experience in software development using Java, J2EE, and Web services.
SRINIVASA.RAO.ARAVILLI@HP.COM

UDDI defines the following major data structures, along with two sets of APIs, to publish and discover Web services:

- ***businessEntity:*** Describes a business or other organization that typically provides Web services
- ***businessService:*** Describes a collection of related Web services offered by an organization described by a businessEntity
- ***bindingTemplate:*** Describes the technical information necessary to use a particular Web service
- ***tModel:*** Describes a "technical model" representing a reusable concept, such as a Web service type, a protocol used by Web services, or a category system

Why JAXM?

JAXM enables applications to send and receive document-oriented XML messages using a pure Java API. JAXM implements SOAP 1.1 with Attachments messaging so developers can focus on building, sending, receiving, and decomposing messages for their applications instead of programming low-level XML communications routines (see "XML Messaging with JAXM," *XML Journal*, Vol. 3, issue 3).

Development of a UDDI Registry

This section explains how to develop a private UDDI registry, illustrating a tModel data structure using the following publish and inquiry APIs:

- ***get_authToken:*** Used to obtain an authentication token. Authentication tokens are opaque values required for all publish API calls; they're optional for inquiry APIs
- ***save_tModel:*** Used to add or update one or more registered tModel elements
- ***find_tModel:*** Used for locating a list of tModel entries that match a set of specific criteria

To develop the UDDI registry, follow these steps:

1. Analyze the UDDI data structures
2. Create the database schema for the data structures
3. Develop the servlets for inquiry and publish requests
4. Deploy the servlets
5. Test the private registry using client applications

Analyze the UDDI Data Structures

UDDI defines the XML schemas for each data structure. The tModel schema is shown in Listing 1 (listings for this article are available for download at www.sys-con.com/webservices/sourcec.cfm).

tModel consists of a key, a name, and optional elements such as URL, category-

Bag, and IdentifierBag, and it may have zero or more descriptions.

tModelKey is the unique key for a tModel structure. Service providers pass an empty tModel key while publishing the Web services, and UDDI registries generate a Universal Unique Identifier (UUID) value using one of the UUID algorithms (www.opennc.org/onlinepubs/9629399/apdxa.htm) and prefix it with a URN qualifier in the format "uuid:", followed by the UUID value. This will be assigned to the tModelKey. Operator is the name of the operator who provides the UDDI registry.

Create the Database Schema for the Data Structures

After analysis of the data structure, one design option is to represent the tModel name and key in one table (as shown below), the tModel descriptions in another table, and categoryBag and identifierBag in different tables.

```
CREATE TABLE TModel (
tModelKey VARCHAR(255)  NOT NULL,
name VARCHAR(255)  NOT NULL,
datestamp TIMESTAMP NOT NULL,
authorizedName VARCHAR (255) NOT NULL,
  CONSTRAINT PK_TM PRIMARY KEY
(tModelKey)
);
```

Develop the Servlets for Inquiry and Publish Requests

UDDI defines the APIs to publish and discover Web services. Develop the servlets for each type of request, namely InquiryServlet and PublishServlet, in order to process the inquiry (e.g., find_tModel) and publish the APIs (e.g., save_tModel).

PublishServlet

PublishServlet handles the following UDDI requests:

- ***get_AuthToken:*** Authentication request
- ***save_tModel:*** Publish request

UDDI_PublishServlet extends the JAXM Servlet and implements the ReqRespListener interface in order to receive and send SOAP messages. This servlet receives a UDDI publish request as a SOAP message and then passes the SOAP message to onMessage () method.

onMessage(SOAPMessage message) is the main entry and exit point for processing the SOAP messages. This method receives the UDDI request as a JAXM SOAP message and returns the SOAP message to the client.

Processing Authentication Requests

The PublishServlet receives the following SOAP message for the get_authToken API:

```
<Envelope
xmlns="http://schemas.xmlsoap.org/soap/env
elope/">
<Body>
<get_authToken  generic="2.0"
xmlns="urn:uddi-org:api_v2"
userID="usename" cred="password" />
</Body>
</Envelope>
```

Retrieve the body of the SOAP message using JAXM APIs to find the UDDI request (i.e., get_AuthToken, save_tModel, etc.). Find the other elements and attributes required for processing the UDDI API. For instance, get_AuthToken requires user name and credentials to generate the authtoken. Retrieve the user ID and credentials from the SOAP message and generate the authtoken using your own algorithm as explained in the genAuthToken() method shown in Listing 2. Create a SOAP message with authtoken details and send the response to the client as explained in the sendResponse( ) method shown in Listing 2.

The response SOAP message for the get_authToken API looks like the following:

```
<Envelope xmlns
=http://schemas.xmlsoap.org/soap/enve-
lope/">
<Body>
<authToken generic="2.0" operator="'your-
companyname' " xmlns="urn:uddi-org:api">
<authInfo>authtokenxyz</authInfo>
</authToken>
</Body>
</Envelope>
```

Processing Publish Requests

The PublishServlet receives the following SOAP message for save_tModel and passes this message to onMessage() method:

```
<Envelope
xmlns="http://schemas.xmlsoap.org/soap/env
elope/">
<Body>
<save_tModel generic="2.0"
xmlns="urn:uddi-org:api_v2">
<authInfo>authtokenxyz</authInfo>
<tModel tModelKey="">
<name>Geography</name>
</tModel>
 </save_tModel>
</Body>
</Envelope>
```

Retrieve and validate the authInfo element. Once the authInfo is valid, retrieve the tModel details, such as name, description, and so on, to save the tModel in the registry.

To save the tModel, construct the SQL query, pass the parameters, and execute the query (see Listing 2).

Create a SOAP response with the generated tModelKey and other details, such as name, operator, and so on, and send the response back to the client. The created SOAP response looks like the following:

```
<tModelDetail generic="2.0" opera-
tor="yourcompanyname"
truncated="false" xmlns="urn:uddi-
org:api_v2">
<tModel tModelKey=" uuid:c1acf26d-
9672-4404-9de0-39b756e62abc " opera-
tor="yourcompanyname"
authorizedName="aravilli">
<name>Geography</name>
</tModelDetail>
```

InquiryServlet

InquiryServlet handles the inquiry API requests only. It extends the JAXMServlet and implements the ReqRespListener interface to receive and send SOAP messages. It receives a SOAP message then passes it to onMessage () method. The received SOAP message for the find_tModel is as follows:

```
<Body>
<find_tModel generic="2.0"
xmlns="urn:uddi-org:api_v2">
<name>Geography</name>
</find_tModel>
</Body>
```

Retrieve the body of the SOAP message using JAXM APIs to find the UDDI request (it could be find_tModel, find_business, etc.). In this case the request is find_ tModel.

To process the find request, construct the SQL query with find details and execute it to fetch results from the registry (see Listing 3).

Create the tModelDetail response as follows, with the result of the find_tModel query as a SOAP message, and send the response back to the client.

```
<tModelDetail generic="2.0"
operator='yourcompanyname'
truncated="false" xmlns="urn:uddi-
org:api_v2">
<tModel tModelKey=" uuid:c1acf26d-9672-
4404-9de0-39b756e62abc" operator='yourcom-
panyname' authorizedName='aravilli'>
<name>Geography</name>
</tModelDetail>
```

Deploy the Servlets

Deploy these servlets in any servlet container with URL mappings (/uddi/in quire = InquiryServlet, /uddi/publish = PublishServ let).

Test the Private Registry Using Client Applications

The UDDI registry can be tested with the following:

- ***GUI tools***
- ***Client applications:*** UDDI4J and JAXR

To test the registry, publish various tModels using the publish URL (http:// webserver-host:port/uddi/publish) and discover the tModel using the inquiry URL (http://web-serverhost:port/uddi/in quire) using UDDI4J or JAXR library.

Summary

I used the Java servlets, HSQL database, and JAXM client libraries to develop a private UDDI registry. The UDDI specification doesn't dictate implementation details; it defines an XML-based data model and a set of SOAP APIs to access and manipulate that data model. The SOAP APIs define the behavior a UDDI registry must exhibit. A UDDI implementation can be built using any language or any directory structure, as long as it conforms to the specified behavior.

Resources

- *Universal Description, Discovery, and Integration:* www.uddi.org
- *Universal Unique Identifier:* www.opennc.org/onlinepubs/9629399/apdxa.htm
- *HSQL Database Engine Project:* http://hsq ldb.sourceforge.net/
- *Java:* http://java.sun.com
- *Simple Object Access Protocol 1.1 (SOAP):* www.w3.org/TR/SOAP
- *Java API for XML Messaging:* http://java.sun.com/xml/jaxm
- *Java API for XML Registries:* http://java.sun.com/xml/jaxr
- *UDDI4J:* www.uddi4j.org

Boundaryless Information Flow

The expanding role of Web services

Web services fits into the operational model that The Open Group calls Boundaryless Information Flow – the secure, reliable, and timely flow of information throughout and between enterprises. At our recent conference, chief officers from enterprises that use Web services in their business, including the business of government, recounted the role of Web services in achieving Boundaryless Information Flow. The tales of successes and challenges from McGraw-Hill, MasterCard, and the federal government offered surprises, affirmed a few fears, and highlighted specific victories in the world of Web services.

McGraw-Hill

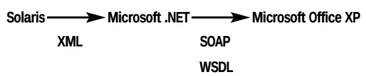
"McGraw-Hill is all about brands, people, and content," said Jim King, McGraw-Hill's chief information officer, as he introduced examples of how the company uses Web services to promote its brands, connect people, and deliver content. As a vertical information provider serving the top industries in the world – education, healthcare, finance, and construction – McGraw-Hill has gone beyond today's best practices in Web services and is now laying the foundation for the future of Web services. Let's look first at what they're doing today, and then push into the future.

The overarching objective of McGraw-Hill's Web services is achieving what King calls "content and context transparency." One of the first questions McGraw-Hill's Web services team asks themselves is, "What is the user's context?". That is, what is the consumption method, or what is the context in which they need McGraw-Hill information? King emphasized that a media organization such as McGraw-Hill gains credibility based on editorial context – and context management can be a nightmare! The act of mapping context to content entails a keen understanding of the workflow of an industry and the people in it.

Transparency must be embedded in the flow of information and the flow of data. In the flow of information, content is delivered within a domain context, and integration into an application services context is transparent to the user. In the flow of data, content within a functional context is transparent to the processes that manipulate that data. The latter point is contingent on interoperability, which is the cornerstone of Boundaryless Information Flow. King discussed two specific use cases involving financial services and construction.

McGraw-Hill's business unit Standard & Poor's has built a set of about 60 Web services, which the company characterizes as encapsulations of content delivery to end-user devices and applications. The number makes sense in light of the comprehensive nature of S&P. The financial services include securities information and evaluations, ratings services, risk management and assessment tools, equity analysis, fund services, financial reporting, value consulting, and much more.

The S&P Web services rely heavily on an alliance with Sun Microsystems, but involve a mixture of technologies designed to use Smart Tags and Microsoft XP to deliver content directly to the customer's desktop:



King continued by noting that the complexity of Web services for the construction industry "is more indicative of why we need all this transparency." McGraw-Hill links facility owners and other key players with the project and certain products. The company currently represents more than 600,000 construction projects in the United States alone. And as he enumerated what types of content support them, King said that in paper form, "there are hundreds of pounds of information per project."

The workflow in the industry between the data and what people in the industry do with the data concurrently engages multiple applications, consumers of information, and products. At any given time, the people who need the data include the facility owner, lead architect, project engineer, general contractor, building parts suppliers, facility manager, and service providers. The primary applications in play are financial, CADD, document management, project management, estimating, and scheduling. McGraw-Hill's (Sweets.com) products number about 61,000, including specifications, downloadable CAD libraries, streamlined contact with local manufacturers' reps, and much more.

Author Bio

Allen Brown is the prresident and CEO of The Open Group LLC. Before joining The Open Group, Brown was a consultant, specializing in start-up and turn-around companies. Prior to this, he held a senior financial position in Unilever Computer Services Limited. Brown has an MBA from the London Business School.

As King noted, "Our industries are hugely complex. They are not transparent. They are not integrated. They are certainly not interoperable. But the winners among the vertical information providers – they are the companies that can get this workflow right and influence the providers of technology to help us map that technology and our content into this workflow."

In trying to create an architecture that serves that goal for its construction market, McGraw-Hill forged an alliance with Microsoft. King asserted that Microsoft's integrated approach seemed up to the task of achieving transparency in Web services for an industry with 1.2 million different businesses. He feels that, in this situation, Microsoft .NET keeps the customer focus on the content, and not how they got it.

King's explanation engendered interesting discussion in terms of The Open Group's vision of Boundaryless Information Flow. The Open Group has an unwavering commitment to work with suppliers, consortia, and standards bodies to develop consensus and facilitate interoperability – to evolve and integrate specifications and open-source technologies. Knowing this, King engaged the audience head-on in a discussion of proprietary Web services solutions versus others. This invariably led to one of the times when conference presentations "affirmed a few fears." In short, the industry needs more open standards that address the issues of security, reliability, and timeliness in the Web services world.

This is one reason why The Open Group took the unprecedented step of bringing together leaders of seven consortia to spotlight how they are contributing to the body of open standards for Web services and to ascertain areas of potential collaboration (see sidebar).

King also teased the audience with a glimpse of what McGraw-Hill sees as the future of Web services: nanoservices. So as vendors contemplate ways of evolving Web services, they should keep in mind that at least one large customer is looking at extending Web services to communication with machines inside the body.

MasterCard

Simon Pugh, vice president of Standards & Infrastructure, MasterCard International, noted that e-commerce now represents 3 to 4% of MasterCard's business, and this is becoming an extremely important new channel. Inhibitors to buying on the Internet are the length of time involved and concerns about security and privacy. Addressing those issues will support MasterCard's e-business strategy to take credit card use into new areas in terms of both transaction size and type.

Standards are key to the solution. Pugh

Boundaryless Information Flow: The Role of Web Services

In a rare display of unified focus, leaders of seven major IT consortia came together at The Open Group's Web services conference in July. "Boundaryless Information Flow: The Role of Web Services" gave executives a chance to spotlight how their organizations contribute to the body of open standards for Web services and to ascertain areas of potential collaboration. A central message for many was the royalty-free nature of their specifications.

Allen Brown, president and CEO of The Open Group, described Boundaryless Information Flow, the framework for specific work areas of Web services, as something to be achieved through global interoperability in a secure, reliable, and timely manner. He outlined how the consortia present might work together to meet customer needs, such as those described by McGraw-Hill, MasterCard, and federal government executives at the conference, and grow the market for Web services. A fundamental element is using, not duplicating, each other's area of expertise. For example, The Open Group brings skills in business process scenarios, integration, certification, and testing.

Patrick Gannon, president and CEO of OASIS, covered his organization's main ebXML concepts in his address "The Building Blocks for eBusiness Web Services." He looked at business processes, business messages, trading partner agreements, and business service interfaces, all expressed in XML; Transport and Routing Layer, which moves the actual XML data between trading partners; and Registry/Repository, which provides a "container" for process models, vocabularies, and partner profiles.

Steve Bratt, COO of the World Wide Web Consortium, described the next steps of the W3C's groups with responsibilities to Web services. These include Architecture (completion of the architecture definition and use cases); XML Protocol (SOAP 1.2 to Recommendation, which should happen late 2002 or early 2003); and WSDL (issues found against WSDL 1.1, design a binding to SOAP version 1.2, and develop an RDF mapping).

Winston Bumpus, president, Distribution Management Task Force, noted that DMTF sees Web-Based Enterprise Management (WBEM) as the forefather of Web services and stressed the need to go beyond standardization in Web services. Active involvement in the implementation of standards is vital, with commitment to compliance testing and certification. In its work, DMTF emphasizes the importance of Web services infrastructure and the interdependence of its components, and the collaboration necessary to develop and promote compliance.

Richard Mark Soley, chairman and CEO, Object Management Group, focused on Model-Driven Architecture. MDA has UML, MOF (Meta-Object Facility), and CWM (Common Warehouse Metamodel) at its heart, using Web services, CORBA, Java, .NET, and XMI/XML to support a range of services.

Carl Reed, executive director of the OGC Specification Program, OpenGIS Consortium, described the work on the OGC Web Services 1.2 specification. This includes Sensor Web capabilities, Vector Data Feature production, Imagery production, and Common Architecture Enhancements. Web services for the mobile domain cover mobile-phone capabilities such as a navigation service find "nearest" service.

George Siegle, past chairman, OAG, provided insights on OAG's deliverables, namely, applications architecture, collaboration definitions in UML, XML messages defined in prose, XML messages expressed in XSD, data dictionary, and using industry-standard frameworks to avoid duplication.

asserted his concern that the standards required to support Web services are not mature, particularly in the area of security. Other issues are a perceived lack of stability in the architectures, the need for vendor-neutral tools, and the costs of technology change.

MasterCard does, however, see expanding potential for the use of Web services in many areas, one of which is their ATM Locator, which enables customers to locate the nearest MasterCard ATM from a device such as a WAP phone or a PDA.

It is also working on a Secure Payment Application, an initiative for Internet transactions that provides a guaranteed payment to the merchant. The consumer is authenticated for the transaction by the issuer, and a token (UCAF – Universal Cardholder Authentication Field) is provided to the merchant, who then submits the populated UCAF to receive the payment guarantee. This application will be XML based and is addressed to the URL of the authentication mechanism of the customer's bank. As a final note, Pugh emphasized that with mobile transactions the situation becomes more complex, and MasterCard is looking to build an authentication Web service.

The Federal Government

In terms of the distance to travel to implement robust Web services, Mark Forman's territory is the largest described by any user presenting at the conference. The Bush Administration's associate director of information technology and e-government, Office of Management and Budget, began with the statement of a daunting IT-related goal: "We're trying to reform the federal government." Forman then carefully outlined how his organization intends to turn $53 billion in Fiscal Year 2003 IT expenditures into benefits for U.S. citizens. His view, expressed strongly in a presentation entitled "The Need for Web Services in the Federal Government," is that Web services comprise a big part of the future IT scheme for federal agencies.

One goal of the Bush Administration is to move the government from being agency centered to being citizen centered. Embedded in that goal, according to Forman, is the vision of "an order of magnitude improvement in the federal government's value to the citizen; with decisions in minutes or hours, not weeks or months." Forman describes the current ROI horror – the reason why big expenditures in technology still have not quickened the pace of decision-making: "We've triple invested and we're not leveraging the business practices." He raised eyebrows by admitting that government agencies have often spent substantial funds to shut down functionalities in the technology they have procured. It has sometimes reflected a conscious avoidance of business practices.

Forman defined e-government as "the use of digital technologies to transform government operations in order to improve effectiveness, efficiency, and service delivery." The aim is not just one of making information available via the Internet – there are thousands of federal Web sites and the government is heavily dependent upon electronic connectivity. The task is essentially one of enterprise resource management. The problem is illustrated by the way in which departments approach it: each does its own, but in fact no one was looking at the total enterprise. At the core of Forman's approach to solving the problem is "simplify and unify."

In essence, "simplify" means adopting simple business practices. At the moment, if a company of any size needs to do business with the federal government, it needs to hire a lawyer or an accountant to navigate the complexities of the system.

"Unify" is just as straightforward. Different arms of the federal government have gathered vast quantities of information that they have not shared with each other, primarily because they had no efficient mechanism to do so. As a consequence, organizations and individuals have been burdened with submitting the same information many times. An inherent part of the solution is that the government, like any business, has to focus on key customer segments – citizens, businesses, intergovernmental contacts, and government employees. To quantify the problem, U.S. businesses alone spend 7.7 billion staff hours a year sending information to the government!

Key technology trends that the federal government is tracking to support the "simplify and unify" strategy are, among others, increasing broadband content and transactional interoperability among government, industry, and individuals; and identifying commodity transaction components that facilitate increasingly agile integration. A central trend Forman hit on was "looking for Web services to become business services."

The federal government defines Web services as Web-accessible automated transactions that are integrated into one or more business processes. They have two main attributes. First, Web services allow the government to build business functionality. Second, they are generally invoked through Web service interfaces, clearly leveraging open standards. A Web service, said Forman, is not a complete solution but a component that contributes to the construction of a solution.

He cited two key opportunities that arise through the use of Web services: accelerating cycle time and enterprise modernization. If Web services can be coupled and scaled in a realistic manner, the federal government will take a giant step toward achieving the Bush Administration's stated goal.

> "Do you want to be one of those people who changes and revolutionizes the way business is done?"

Forman also identified what sort of Web services opportunities exist right now:

- ***Services to citizens:*** Many require hooking up with basic GIS information
  –*Disaster management*: Location of assets, predictive modeling results, and availability of hospital beds
- ***Support delivery of services***
  –*Strategic planning:* Access to capability, decision support, data availability, and analysis
- ***Internal operations and infrastructure***
  –*Financial management*: Debt collection, payment processing, collection, and reporting

He saw the following fundamentals for success in applying Web services:

- ***Identify*** common functions, interdependencies, and interrelationships, and evaluate barriers to information sharing.
- ***Implement*** Web services in a way that addresses both the opportunities and risks of a "networked" environment; security becomes a key element.
- ***Leverage*** technologies to achieve benefits of interoperability while protecting societal values of privacy, civil liberties, and intellectual property rights, among other things.

In conclusion, Forman turned to the burning question of how to leverage Web services quickly, and then highlighted the following issues that need to be resolved:

- Should we have our own UDDI server in the federal government? Who should own a UDDI server and enlist WSDL?
- What shared Web-accessible transaction components are available from whom (e.g., search, PKI-related services, patching services)?
- Which agency has a business model and can write the business case?
- Are Web services supply or demand driven, that is, provided when there is enough demand, or ahead of demand in order to achieve changes?

The Open Group Point of View

Web services play a central role in fulfilling the mission of Boundaryless Information Flow, so the same fundamental elements must be present in both – security, timeliness, and reliability. Whether we're talking about enterprises like McGraw-Hill and MasterCard or the agencies of the federal government, none of these objectives will be realized in a customer-centric way without collaboration between buyers and suppliers.

The sense of urgency that energizes that collaboration – that makes things happen at Internet speed – comes from committing to the vision. In short, we achieve Boundaryless Information Flow in our information infrastructures when all parties involved have action plans and timelines to achieve it. Ultimately, it comes down to a matter of choice. Do you want to be one of those people who changes and revolutionizes the way business is done? The Open Group does and hopes others will, too.

INTEROPERABLE SOAP

Solving the Microsoft/Java problem

Written by Keith Shaffer

The purpose of SOAP is to enable transparent – language- and platform-independent – access to services. Until recently, getting Microsoft- and Java-based SOAP applications to work together has been difficult.

Using the newest SOAP implementations, this article demonstrates how to use SOAP's RPC messaging between Microsoft ASPs and Java JSPs and servlets.

As the World Wide Web Consortium (W3C) works to stabilize its recommendation for SOAP and SOAP-related technologies, transparent access to services continues to be problematic. Creating a SOAP application for a single platform, like just Microsoft or just Java, usually works well. However, getting it to work across platforms, like across Microsoft and Java, can be challenging. Often, to avoid cross-platform interoperabilities, developers choose to write low-level code that creates and parses the XML in SOAP messages.

Another solution that allows developers to concentrate on using services is to use SOAP's RPC-oriented messages. RPC-oriented messages are much like traditional function or method calls, where the sender passes data into a method using arguments, and receives data back in return. RPC-oriented messages are well suited for COM or Java objects communicating via SOAP. This article demonstrates how to create RPC-based, interoperable SOAP applications using the latest SOAP implementations from Microsoft (SOAP Toolkit 2.0 SP2) and for Java (Apache's Axis beta 3). It examines using a Java application as a SOAP client with a Microsoft SOAP server and vice versa.

AUTHOR BIO:

Keith Shaffer spent 12 years in the aerospace industry, most in the R&D department of Hughes aircraft Compnay, then began training and consulting on a range of technologies. For the last several years he has specialied in object-oriented analysis and design, Java, and XML technologies.
KSHAFFER@TELOCITY.COM

Interoperability Problems

Two primary complications make it difficult to create interoperable Microsoft- and Java-based SOAP applications. The use (or non-use in the case of some Java-based implementations) of a Web Services Description Language (WSDL) file is one complication. The other is related to the structure of SOAP messages. When using RPC-oriented messages, the SOAP middleware represents an operation's input and return data as XML so that neither the client nor the server has to work directly with XML. Sometimes the messages used by Microsoft and Java SOAP middleware are incompatible with each other.

WSDL and SOAP

A WSDL file describes (in XML format) the services (or set of operations) offered by a server.. Microsoft SOAP relies on WSDL. In contrast, many Java implementations do not require it, and some may not even support it.

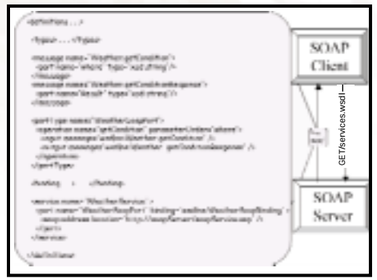


FIGURE 1 | **Client receiving a WSDL file**

A WSDL file typically has five sections: types, message, portType, binding, and service (see Figure 1). A *portType* section lists the operations available to the client. There is one named getCondition. The *message* section maps at most one input and one output message to an operation. In Figure 1, the messages Weather.getCondition and Weather.getConditionResponse are the input and output, respectively, for operation getCondition. These message definitions specify the SOAP message structure, including the name and data types of the arguments. Notice that the Weather.getCondition message has one string argument named where. A *service* section defines how a SOAP client should handle ports, or groups of operations. The port usually specifies the URL of a SOAP request handler, like http://soapServer/soapService.asp in this example. The remaining two sections, *binding* and *types*, are less important for interoperability issues, and Figure 1 does not show their details.

Since many SOAP implementations provide

tools to automatically generate a WSDL file from a COM object or Java class files, a developer may seldom have to completely write it. However, a developer of SOAP client code may have to know the relationship between a WSDL service, port, and operation. It is possible for a single WSDL file to define multiple services. Furthermore, each service may define many ports. Lastly, each port lists a group of operations handled by a SOAP server.

The problem with WSDL and SOAP is that implementers, particularly in the Java camp, have realized that WSDL is not a necessary part of SOAP. In addition, the W3C has not included the WSDL specification in their working draft of SOAP 1.2, and has not embraced using WSDL as a standard way of describing Web services. Since Microsoft's SOAP implementation is dependent on WSDL (in fact, even Microsoft's SOAP server, alone, needs it), using it with an implementation of SOAP that does not use WSDL is problematic. So for interoperable solutions, the Java SOAP server should supply a WSDL file.

SOAP Messages

SOAP messages vary depending on whether a SOAP implementation uses a WSDL file. The WSDL file specifies the structure of SOAP request and response messages. For SOAP implementations that do not require the WSDL, the structure of the messages is implied, or is manipulated through code.

While Java implementations may vary, the approach used by Axis is representative. A SOAP client calls the operation getCondition, which takes a string argument, a city, and returns the current weather condition.

The Microsoft request message has the name of the operation in both the HTTP header (the value of SOAPAction) and in the Body element of the SOAP envelope. Also, the argument for getCondition is labeled *where*. WSDL specifies the value of SOAPAction (in the binding section) and the labels for the arguments (in the message section). If the request message does not have a SOAPAction and correct argument labels, Microsoft's SOAP handler on the server will fail.

In the default (implied) request message from Axis, SOAPAction in the HTTP header does not have a value, and the argument for getCondition is labeled arg0, instead of where. The Axis SOAP handler on the server only looks for the name of the operation in the SOAP envelope (not the HTTP header), and it doesn't care what the arguments of the operation are named, just their presence and order. A WSDL file is not needed for this message structure, since the SOAP client code provides the name of the operation and the values for the arguments. The argument and return types are xsd:string. For Axis, data type information that is normally in a WSDL is instead in the SOAP envelopes.

In summary, achieving interoperable SOAP is complicated because:

- ***Microsoft client to Java server:***
  – Java server correctly handles request message: it ignores the SOAPAction in the HTTP header and cares what the arguments of the operation are called. Also, it does not need to get the data types of the arguments from the message.
  – MS client correctly handles response message: it ignores the type information in the response. The problem: the Java Server may not produce a WSDL that the MS client needs.
- ***Java client to Microsoft server:*** Microsoft server does not handle the request message because it is missing a value for SOAPAction, and the name of the argument, arg0, is wrong. A Java client might not use WSDL and does not know the data type of the response.

Interoperability Solutions

Fortunately, interoperability between Microsoft- and Java-based SOAP implementations is possible. In fact, with Java products like Axis (first beta release on March 15, 2002) it is becoming easier. The solution to using a Microsoft SOAP client with a Java-based SOAP server requires the creation of a WSDL file for the client. There are at least two solutions to using a Java-based SOAP client with a Microsoft SOAP server. The first approach is to have the client read a WSDL to identify how to structure the send message. The second, less attractive, approach is to hard code details, like the argument names and types. While this approach is tedious and less maintainable, it may be necessary in some cases, like when a Java-based SOAP implementation cannot parse a WSDL file.

Microsoft SOAP Client to Java Server

Using a Microsoft SOAP client with a Java-based SOAP server is straightforward as long as the SOAP server has a WSDL file. Consider the scenario where a user submits form data from a Web page to an ASP page. The ASP page then makes a SOAP request to an Axis Servlet (see Figure 2).

The code in the ASP page has two parts. The first part mainly sets which WSDL file is to be used, depending on whether this page should use a Microsoft or Java SOAP server. The second part creates a SoapClient and initializes it with a WSDL file. The call to ClientProperty enables "server safe" loading of the WSDL file. Once the SOAP client initializes the WSDL file, using mssoapinit, the client can then invoke any operation in the first port of the first service of the WSDL file. Recall that in WSDL terminology, a port is a group of operations, a service is a group of ports, and a WSDL file can specify more than one service. So while it is also possible to name a specific service and port in the mssoapinit call, a SoapClient object represents only one port (group of operations) of one service. The line inside the try-block calls the getCondition operation, passing in the value of the city obtained from the HTML form in the Web page.

Listing 1 is from the file weather_ms Client.asp (source code for this article is available at www.sys-con.com/webservices/sourcec.cfm). Notice that this ASP code works whether it sends a request to either a Microsoft or a Java SOAP server. The trick when using a servlet is to create a WSDL file that this client code can use. Fortunately, Axis can automatically generate a WSDL file from Java source code. Point your browser to the URL of the SOAP request handler, adding a WSDL to the URL. A request handler is simply a normal Java source file that is given a .jws, instead of the normal .java, extension. Axis also includes command-line utility to generate a WSDL from a Java source file. IBM, who along with Microsoft authored the WSDL specification, developed this utility, and it is available independent of Axis.

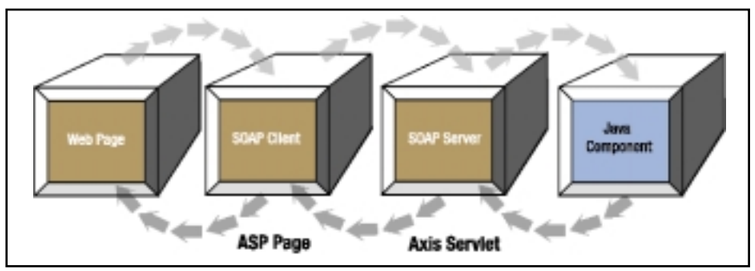


FIGURE 2 | **ASP page making request to Axis servlet**

Unfortunately, the generated WSDL file does not always work without modification. It will work with ASP running under IIS 5.0, but you will probably have to modify it to work with ASP for PWS 4.0, as shown below. In the binding section of the WSDL, there is an element named operation in the http://schemas.xmlsoap.org/wsdl/soap/namespace. The value of this element's soapAction attribute goes in the HTTP header of the SOAP request. The generated WSDL sets the value of the soapAction to the empty string. The ASP SOAP client running under PWS expects soapAction to have a value; it can be nearly anything since the Java SOAP server does use it.

The minor edit to the WSDL file, javaServer\WeatherService.wsdl, generated by Axis, looks like this:

```
<wsdlsoap:operation
soapAction="thisIsForPWS" />
```

Component Design

The component that provides the current weather condition, whether written in Java or Visual Basic, has the same design. It is a class named Weather with just one method, named getCondition.

Component Coding

The Java and Visual Basic code that implement this class have a similar structure, and both use the city name to randomly generate a weather condition.

Looking at the Java source code in Listing 2, notice that the class has two parts. The first defines a list of possible weather conditions and the second is the getCondition method. It initializes a Random object based on the hash code of the city named in the variable where. This code is from the file javaServer\Weather.jws. (Remember that in Axis, Java files have the .jws extension instead of .java. Inside, the Java file has the normal structure.)

Java SOAP Client to Microsoft Server

When a client uses a WSDL file, the JSP code looks much like ASP SOAP client code (discussed earlier). Additionally, the Java client could configure the SOAP service without a WSDL file. With both of these approaches a Web page sends user input to a JSP page, which in turn uses an ASP SOAP server to satisfy the request. The SOAP server uses a COM object written in Visual Basic to produce the return data. Microsoft's SOAP Toolkit includes a utility that generates the ASP SOAP Server page and the WSDL from a COM object (see Figure 3).

The JSP page, which uses a WSDL, has two sections. Skipping the page import statements, for now, the first section sets up the WSDL-related parameters, depending on whether it uses a Java or Microsoft SOAP server. The second section creates a SOAP service object based on a service named WSDL file. Remember that WSDL can specify more than one service. From the service object, the code next creates a call object that represents a single operation. Since a WSDL file groups operations into ports, the create-

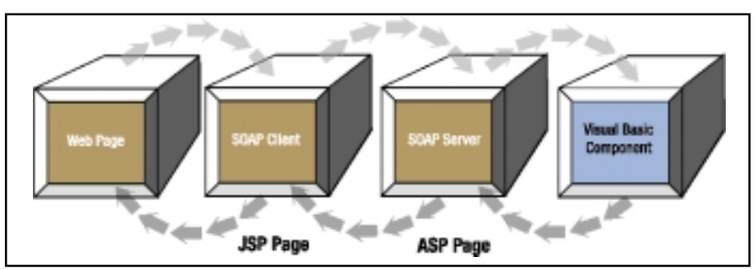


FIGURE 3 | **Converting between JSP and ASP pages**

FIGURE 4 | **Index.html loaded into browser**

FIGURE 5 | **Sample Web page**

Call method must specify which port the operation is in. The last call simply invokes the getCondition operation, passing the value for the city and getting the current in return. Listing 3 is from the file weather_javaClient.jsp.

Most of Listing 3, though written for Axis's SOAP implementation, applies to other Java-based SOAP implementations. Those familiar with Java can tell from the import statements of the first five lines that nothing in the code is unique to Axis. The classes in the import statements beginning with java or javax are standard or standard extension Java classes, and all future Java-based SOAP implementations should use them.

The logic for the Weather class in Visual Basic is the same as the Java code described in the previous section. It has an additional method to generate a hash code from the city.

Example Application

To start the sample applications, load index.html into a browser. It should look like Figure 4. Enter a city, configure the SOAP client and server, and click the "Investigate" button.

If you enter Arlington and choose the JSP client and the ASP Server, the resulting Web page should look like Figure 5. Repeat the process, using other names and other SOAP applications.

Application Setup

System Requirements

The example in this article requires several technologies. In addition to Microsoft's Internet Information Server (IIS) or Personal Web Server (PWS), and a browser such as Internet Explorer, the samples for this article use the following, freely available, products.

- Microsoft SOAP Toolkit 2.0 SP2 (http://msdn.microsoft.com/downloads/default.asp?URL=/code/sample.asp?url=/msdn-files/027/001/580/msdncomposite-doc.xml)
- Visual Basic Run-Time (http://download.microsoft.com/download/vb60pro/Redist/sp4/win98/EN-US/VBRun60sp4.exe)
- Sun Java 2 SDK, Standard Edition v1.4 (http://java.sun.com/j2se/1.4/download.html)
- Apache Jakarta Tomcat v4.0 (http://jakarta.apache.org/builds/jakarta-tomcat-4.0/release/v4.0.3/bin/jakarta-tomcat-4.0.4-LE-jdk14.zip)
- Apache Axis beta 3 (http://xml.apache.org/axis)

To install the Microsoft SOAP Toolkit, the Visual Basic Run-Time, and the Java 2 SDK, double-click on each of the downloaded files and accept the default installation.

Here is a simple procedure for setting up Tomcat and Axis:

- Persistently (may require a reboot) set the environment variable: JAVA_HOME=C:\j2sdk1.4.0_01.
- Unzip the Tomcat and Axis downloads (jakarta-tomcat-4.0.4-LE-jdk14.zip and xml-axis-beta3-bin.zip) into a folder of your choice; the examples in this article use C:\tomcat_ axis.
- Copy the contents of C:\tomcat_axis\axis-1_0\lib into C:\tomcat_axis\jakarta-tomcat-4.0.4-LE-jdk14\lib.

Once your system is set up, as described in the System Requirements Section, use the following steps to set up the application.

- Unzip the iSoap.zip file to C:\tomcat_axis\jakarta-tomcat-4.0.4-LE-jdk14\webapps.
- Set up Web sharing on C:\tomcat_axis\jakarta-tomcat-4.0.4-LE-jdk14\webapps\iSoap:
  – Right-click on the folder.
  – Select Properties
  – In the Web Sharing tab, click on Share this folder.
  – Add the alias iSoap.
- Register WeatherProject.dll by double-clicking registerDll.bat in C:\tomcat_axis\jakarta-tomcat-4.0.4-LE-jdk14\webapps\iSoap\ms Server. (For Windows 98, you may have to type [or copy and paste] regsvr32 C:\tomcat_axis\jakarta-tomcat-4.0.4-LE-jdk14\ webappsiSoap\msS erver \WeatherProject. dll into the Wind ows Start/Run… dialog box.)
- Start Tomcat by double-clicking C:\tomcat_axis\jakarta-tomcat-4.0.4-LE-jdk14\bin\startup.bat.
- Load C:\tomcat_axis\jakarta-tomcat-4.0.4-LE-jdk14\webapps\iSoap\index.html into a browser.

Conclusion

Interoperability between Microsoft- and Java-based implementations of SOAP continues to be difficult, but possible. Until the W3C produces a stable SOAP recommendation, it is not surprising that there are difficulties. The interoperability issues stem mostly from the use (or non-use) of the WSDL, as well as the differences in the SOAP messages. Microsoft SOAP uses WSDL, most Java-based SOAP does not need it and the W3C has decided not to include WSDL with the SOAP recommendation. For RPC-oriented messages over HTTP, Microsoft SOAP servers expect request messages to have operation information in the HTTP header and use specific names for operation arguments. Java-based SOAP servers do not use HTTP header information, and do not care what name is used for the operation arguments. Java-based SOAP clients and servers put data type information into the request and response messages; Microsoft clients and servers do not.

Solving these interoperability problems is possible. Creating a WSDL for Java-based SOAP implementations solves many of the problems. The samples in this article used a Java-based SOAP implementation from Apache, named Axis (in beta at the time of this writing), which does support WSDL. While the samples used Axis, they are representative of the kind of solution required for any Java-based SOAP implementation.

A Java-based SOAP implementation is quite flexible. It does not require WSDL, and a developer has a lot of control in structuring the client calls. The Microsoft-based implementation, while not as flexible because it requires the WSDL, is a solid implementation that effectively uses WSDL to simplify a developer's task in writing client-side code.

Listing 1

```
<%@ LANGUAGE = JScript %>

<%
soapServer = Request("soapServer");
if (soapServer.count > 0 && soapServer == "java")
{
   title = "ASP to Servlet Interaction";
   wsdl =
"http://localhost:8080/iSoap/javaServer/WeatherService.wsdl";

   useJavaServer = "checked='checked'";
   useMsServer = "";
}
else
{
   soapServer = "ms";
   title = "ASP to ASP SOAP Interaction";
   wsdl =
"http://localhost/iSoap/msServer/WeatherService.WSDL";

   useJavaServer = "";
   useMsServer = "checked='checked'";
}

%>
```



```
<%
where = Request("city");
condition = "";
if (where.count > 0)
{
   service =
Server.CreateObject("MSSOAP.SoapClient")

service.ClientProperty("ServerHTTPRequest")
= true
   service.mssoapinit(wsdl, "", "", "");
   try
   {
      condition =
service.getCondition(where);
   }
   catch (exc)
   {
      condition = exc.description;
   }
}
else
{
   where = "";
}
%>
```

Listing 2

```
import java.util.Random;

public class Weather
{
 private static String[] conditions =
 {
  "Sunny", "Partly cloudy",
  "Partly sunny", "Cloudy", "Precipitation"
 };

 public String getCondition(String where)
 {
  Random weather = new
      Random(where.hashCode());
  int w =
      weather.nextInt(conditions.length);

  return conditions[w];
 }
}
```

Listing 3

```
<%@ page import="java.net.URL" %>
<%@ page import="javax.xml.namespace.QName"
%>
<%@ page import="javax.xml.rpc.Call" %>
<%@ page import="javax.xml.rpc.Service" %>
<%@ page
import="javax.xml.rpc.ServiceFactory" %>

<%
String title;
String serviceName;
QName portName;
String targetNs;
String wsdl;
String useJavaServer;
String useMsServer;

String soapServer =
     request.getParameter("soapServer");
if (soapServer != null &&
soapServer.equals("java"))
{
   title = "JSP to Servlet Interaction";
   serviceName = "WeatherService";
   portName = new QName("Weather");
   wsdl
="http://localhost:8080/iSoap/javaServer/We
     atherService.wsdl";
   targetNs = "http://localhost:8080/" +

"iSoap/javaServer/Weather.jws/iSoap/javaSer
     ver/Weather.jws";

   useJavaServer = "checked='checked'";
   useMsServer = "";
}
else
{
   title = "JSP to ASP SOAP Interaction";
   serviceName = "WeatherService";
   portName = new QName("WeatherSoapPort");
   wsdl =
"http://localhost/iSoap/msServer/WeatherSer
vice.WSDL";
   targetNs = "http://tempuri.org/wsdl/";

   useJavaServer = "";
   useMsServer = "checked='checked'";
}
%>

<%
String where =
     request.getParameter("city");
String condition = "";
if (where != null)
{
   try
   {
      ServiceFactory sf =
     ServiceFactory.newInstance();
      Service service = sf.createService(
         new URL(wsdl), new QName(target
     Ns, serviceName));
      Call getCondition =
         service.createCall(portName, new
QName("getCondition"));

      condition = (String)
getCondition.invoke(new Object[] {where});
   }
   catch (Exception e)
   {
      condition += e.getMessage();
   }
}
else
{
   where = "";
}
%>
```

Download the code at sys-con.com/webservices

WebServices .NET J2EE XML JOURNAL

PUBLISHER, PRESIDENT, AND CEO
Fuat A. Kircaali fuat@sys-con.com

COO / CFO
Mark Harabedian mark@sys-con.com

VP, BUSINESS DEVELOPMENT
Grisha Davida grisha@sys-con.com

ADVERTISING
SENIOR VP, SALES & MARKETING
Carmen Gonzalez carmen@sys-con.com
VP, SALES & MARKETING
Miles Silverman miles@sys-con.com
ADVERTISING DIRECTOR
Robyn Forma robyn@sys-con.com
ADVERTISING ACCOUNT MANAGER
Megan Ring-Mussa megan@sys-con.com
ASSOCIATE SALES MANAGERS
Carrie Gebert carrie@sys-con.com
Alisa Catalano alisa@sys-con.com
Kristin Kuhnle kristin@sys-con.com
Leah Hittman leah@sys-con.com

SYS-CON EVENTS
VP, EVENTS
Cathy Walters cathyw@sys-con.com
REGIONAL SALES MANAGERS, EXHIBITS
Michael Pesick michael@sys-con.com
Richard Anderson richard@sys-con.com
CONFERENCE MANAGER
Michael Lynch mike@sys-con.com

CUSTOMER RELATIONS / JDJ STORE
MANAGER, CUSTOMER RELATIONS
Anthony D. Spitzer tony@sys-con.com
CUSTOMER SERVICE REPRESENTATIVE
Margie Downs margie@sys-con.com
MANAGER, JDJSTORE
Rachel McGouran rachel@sys-con.com

WEB SERVICES
VICE PRESIDENT, INFORMATION SYSTEMS
Robert Diamond robert@sys-con.com
WEB DESIGNERS
Stephen Kilmurray stephen@sys-con.com
Christopher Croce chris@sys-con.com
Catalin Stancescu catalin@sys-con.com
ONLINE EDITOR
Lin Goetz lin@sys-con.com

ACCOUNTING
ASSISTANT CONTROLLER
Judith Calnan judith@sys-con.com
ACCOUNTS RECEIVABLE / COLLECTIONS SUPERVISOR
Kerri Von Achen kerri@sys-con.com
ACCOUNTS PAYABLE
Joan LaRose joan@sys-con.com
ACCOUNTING CLERK
Betty White betty@sys-con.com

SUBSCRIPTIONS
SUBSCRIBE@SYS-CON.COM
1-888-303-5282
FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS, PLEASE SEND YOUR LETTERS TO SUBSCRIPTION DEPARTMENT
COVER PRICE: $6.99/ISSUE
DOMESTIC: $69.99/YR (12 ISSUES)
CANADA/MEXICO: $99.99/YR
ALL OTHER COUNTRIES: $129.99/YR
(U.S. BANKS OR MONEY ORDERS)

SYS-CON MEDIA

Web Services in a Wireless World

The challenge of WS2C

Conventional wisdom is a curious thing, especially when applied to a technology as new as Web services. Web services are often thought of as distributed business processes participating in a B2B relationship using Internet protocols and XML-based data standards. In fact, much of the introductory literature on Web services emphasizes that while most modern Web applications are designed to be used by human beings sitting in front of a browser, Web services are supposed to be used by some sort of automated service requester. The examples given in such texts frequently support such assumptions, but is this the only useful scenario?

In a typical B2B relationship, the actual client application logic is likely to be found in the middle tier of a distributed application of some sort. Frequently, but not exclusively, this middle-tier logic is a business object in the form of an EJB or .NET component. Fundamentally, this component's relationship with Web services is likely to be similar to its role with other components in a traditional distributed-object system based on CORBA, COM+, or Java technology. The component could be the client of more than one Web service, and it could be a Web service itself.

However, it becomes increasingly clear to many architects and designers that the business logic exposed by properly designed Web services is often coarser-grained than the logic exposed by so-called "business objects" in the more tightly integrated distributed object systems. In other words, Web services should usually function at a higher, more business-centric level than do middle-tier business objects in distributed Web applications.

Business objects, despite the name, are often not very business-like. To illustrate this concept, imagine some fairly officious "business EJB" carrying out requests to "update customer table" or "debit account." Indeed, this is often the level of functionality exposed by business objects in many distributed business applications. The level of business abstraction in most object-oriented, component-based systems is not very high. This may be a failure in design, but it's nevertheless often left up to the Web components (such as ASPs or JSPs) to coordinate these lower-level requests based on higher-level human instructions. These higher-level human instructions, precipitated at the browser level, function more truly at a real business level. For example, that very low-level business logic I just mentioned might be invoked by very high-level and meaningful human business decisions to do something like "join the book club" or "purchase a new bank service." Thus, it's fair to say that business components in applications based on tightly integrated, object-oriented technologies often function at a finer-grained level than Web services.

Web services function at a higher business level for many reasons. One of the most obvious is the overhead associated with parsing XML for each invocation of a high-level business function. An even more significant reason is the philosophical underpinnings of Web services as compared to highly integrated, object-oriented systems. Web services are typically applied to integration tasks rather than distributed application development tasks. This difference of purpose is somewhat subtle. However, Web services standards such as SOAP, UDDI, and WSDL were developed with much thought about issues closer to the EAI (Enterprise Application Integration) world, rather than distributed application development, and that has made all the difference, as the well-known poet (and erstwhile Web services consultant) Robert Frost once said.

When solving EAI problems, loose coupling is essential. Loose coupling means minimizing the dependencies between systems. People experienced with EAI know that dependencies between autonomous systems, possibly using radically different technologies, are very dangerous. Binary data formats can change, protocols can change, and parameters associated with business logic calls can change. The list of risk factors is endless, and you can never really trust either side to behave properly all the time. So, while tightly coupled distributed applications can be built quickly and can be very efficient (good goals for an application built within the enterprise or a corporate division where the IT department can exercise control), loosely coupled systems can potentially offer faster, easier, and safer integration beyond the firewall, or between any two systems that are based on different underlying technologies. Throwing away all the rhetoric, this is a very compelling value proposition for Web services.

Author Bio

Paul Lipton is a director and technology strategist at Computer Associates International, Inc. (CA). He has been an architect and developer of enterprise systems for more than 20 years and has represented CA in various standards organizations. Paul has spoken at numerous conferences and seminars and has published articles on Web services, Java, .NET, EAI, wireless technology, and distributed systems.
PAUL.LIPTON@CA.COM

Why Wireless?

Why should people interested in Web services also care about wireless? It's beyond the scope of this article to examine the demographics and analyst reports indicating an observable rise in the use of wireless technologies throughout the world. For the sake of discussion, I'll simply say that I believe the widespread use of wireless technology is both inevitable and around the corner, as long as your definition of corners is a couple of years long. In fact, many IT development projects now wisely incorporate some sort of long-term plan to support wireless devices. Certainly, the huge commitment of companies (e.g., Microsoft with its Smartphone technology, based on the compact .NET Framework; and Sun Microsystems with J2ME) illustrate that the software industry is taking wireless very seriously. Wireless companies, such as Qualcomm with their BREW platform, are also pursuing aggressive wireless software platform initiatives.

Unlike the business objects currently implemented in many Web applications, the ideal Web service is granular enough to expose meaningful business functionality. So why wouldn't end users want to take advantage of that business functionality on the many types of wireless devices that they're likely to use in the next few years for at least some of these Web services? Indeed, Web service creators may be driven to respond to this demand for increased accessibility and convenience by making their service available to an equally new generation of wireless end users, in addition to the more traditional service requesters from other systems. However, Web service creators who come from a traditional Web application development environment may find that the world of wireless presents new challenges.

To directly access Web services, end users need an application user interface for their wireless devices, as well as conventional PCs. In the past, the task was simpler. Designers of conventional Web applications experienced a relatively restricted range of client hardware. Their various display and navigational characteristics were fairly small and uniform. Many Web applications could reasonably assume a pixel-addressable color display of fairly high resolution (at least 640x480 pixels) running on a very small number of browsers. In fact, it was often sufficient for developers to ensure that the application worked properly with the various versions of IE and older versions of Netscape. Even the most forward-thinking developers often did little more than extend their development and testing to include the latest Mozilla-based browsers as well as the increasingly popular Opera browser. With a total of four HTML browsers at the most, and mouse-driven navigation of the user interface as the norm

living within a high-resolution color display, development was a challenge that a good Web team could master.

Achieving WS2C

The very diversity of wireless devices currently poses a different sort of challenge for creators of new end user–enabled Web services: Web Service to Consumer, or WS2C. There's an incredible diversity of device hardware, navigation styles, displays, and operating software available for these devices. While certain enterprise applications will be able to mandate the wireless device type for clients, many Web services developers won't have that power over their end users. Hardware designs in the wireless world are far from standardized. Displays vary in color depth and resolution, with the displays of the simplest devices limited to black and white characters instead of color pixels. Navigation can be by many mechanisms, including arrow keys or a pen. Memory and processor speed are very diverse, and the resident software platform can be a thin-client browser or a fat client. In addition, many vendors are vying for a share of the action.

Note that despite all the posturing by vendors, there's truly no market leader in the wireless market. Just the opposite is true, so it's difficult to limit the breadth of support for wireless devices. Furthermore, the diversity of devices, from PDA/phone combinations to unusual form factors and software capabilities, will only increase. Nor is it certain that the new, fatter-client wireless devices based on technologies such as J2ME, BREW, and Smartphone will rapidly replace the hundreds of millions of WAP-enabled phones currently in use. In fact, it may be more appealing in some markets to use less complex phones and their simple browsers. The only safe assumption is that many devices and technologies will continue to fight for market dominance for some time, as both the bandwidth and the usefulness of wireless devices increase.

Thus, those building business-level Web services that are WS2C-enabled over the next year or two will be faced with an interesting challenge. In the past, exposing business functionality over the Web has meant writing a Web component, such as an ASP or JSP, consisting of some fairly static markup (typically HTML) along with logic to coordinate lower-level calls to business objects. Certainly, such an approach could be used to front-end a Web service either by directly accessing the Web service or by using an underlying business object like an EJB. But, does this architecture make the most sense in today's world of extremely diverse clients?

Writing a different Web component with the appropriate markup (WML, XHTML, etc.) for all the different wireless browsers in the market now and in the future would make maintenance quite difficult. For more intelligent devices, it might be sufficient to directly pass the XML derived from the Web service to the wireless device for parsing and display. Thus, the ability to sense the client type and dispatch very different transformations of XML data (or none) to a client becomes important. That's why wireless technologies embedded in server-side middleware like some of the various Java application servers and .NET feature some sort of transformation technology to sense the client type and dynamically convert the content to a format acceptable to the device.

Many wireless technologies offered by middleware vendors require the addition of special markup tags or instructions in their Web components so that transformations to various wireless devices can be done intelligently by the server. This is often a less obvious aspect of committing to a middleware vendor, and the result ties one to that vendor tightly. Since there are no leaders or dominant standards for such technology, it might require significant effort to port the wireless application to another vendor, even if both vendor products are based on the same underlying technology, such as Java.

Architecturally, it may be best to consider placing the wireless support closer to the end users. Transformations at this level can be independent of the underlying middleware. This shields the enterprise from having to commit to a single underlying technology like J2EE or .NET for wireless initiatives. Many decision makers are still hedging their bets between these two competing technologies at many levels, and wisely so.

The corporate portal, which inhabits a tier much closer to end users, is already an important part of any proposed WS2C solution for many enterprises, as it provides the personalization and security necessary to make a Web service truly useful, secure, and accessible to end users. It will also be the essential role of portals to support that same personalization and security on a wide range of wireless devices. Given the diversity, rate of change, and lack of leadership in the wireless industry, wise architects and decision makers incorporating portal technology as part of their infrastructure will need to ensure that their portal has wireless support that is truly open, extensible, and standards- compliant.

Web Services Infrastructure

Building transactional Web services with OASIS BTP

Written by Jim Webber

In part 1 of this article (*WSJ*, Vol. 2, issue 10), you saw how simply BTP toolkits can support the creation of applications that drive transactional Web services with consummate ease. This article covers the other side of the story: how the same technology impacts Web services developers.

In this article, I'll address this aspect and show how BTP can be used to create transaction-aware Web services and how those services can be consumed by transactional applications.

AUTHOR BIO:

Dr. Jim Webber is an architect and Web services fanatic at Arjuna Technologies, a UK-based start-up focusing on transactions and reliable messaging. Previously, Jim led the team that developed the world's first XML-based transactioning system at Hewlett-Packard middleware and continues today working at the code-face and in talking-shops with Web services transaction technology (like BTP and WS-Transaction). JIMWEBBER@HOTMAIL.COM

Transactionalizing Web Services

To transactionalize a Web service with BTP is something of a misnomer, since BTP doesn't deal with transactional Web services per se, choosing instead to partition Web services into two distinct types to enable clear separation of business services and their associated participants.

Business services are similar to client applications in that there is no inherent transactionality associated directly with them – they simply exist to host and expose business logic. On the other hand, the participants associated with business services are essentially business-logic agnostic and deal only with the transactional aspects of service invocations. This is quite useful, since it means that existing services can be given transactional support without necessarily performing any invasive procedures on them. The fact that nontransactional Web services can be given transactionality without having to rebuild the service is a real plus. It also means that transactional and business aspects of a system can be evaluated and implemented independently. With these additional pieces of the puzzle, you can now reshape the global BTP architecture shown in Figure 1.

Figure 1 typifies a logical BTP rollout, showing how the endpoint of each BTP actor fits into the global model. You can see that the services

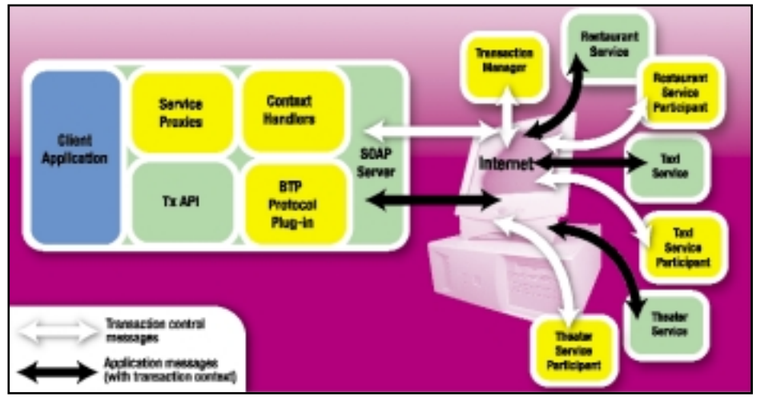


FIGURE 1 **The global BTP architecture**

that expose business logic to the Web are supported by other services, called participants, that deal with the transaction management of those business-oriented services and how, importantly, there is a clean separation between the two kinds of service. There's clearly some overlap, even at this level, since application messages carry BTP contexts whenever service invocations are made within the scope of a transaction. It is here that the business logic and transaction domains begin to collide, albeit gently.

Supporting Infrastructure

For business Web services, most of the interesting work from a transactional perspective happens under the covers. Like the client application, Web services benefits from advances in SOAP server technology that support header processing before the application payload of a SOAP message is delivered. For BTP-aware Web services, you can utilize SOAP header processing to insert and extract BTP contexts on behalf of Web services in a fashion reciprocal to how header processing is performed at the client application side. Since header processing is noninvasive to the service-level business logic, you can see how the impact of making a service transactional with BTP is minimal. Figures 2 and 3 show exactly how the service is supported.

Figure 2 demonstrates what happens when a Web service receives a request. If the request doesn't carry a BTP context, it's simply passed through the incoming context handler to other handlers and will eventually deliver its payload to the service. If, however, the request carries a BTP context, then the context is stripped out of the header of the incoming message and associated with the thread of execution within which the service's work will be executed. To achieve this, the handler resumes the transaction, using elements from the transaction manager part of the API we saw in the first article, which effectively associates (or reassociates, if this isn't the first time the same context has been received) the work performed by the service with a BTP transaction.

When returning from a service invocation, the reverse process occurs, as shown in Figure 3. The message from the service passes through the outgoing context handler, which checks to see if there is a transaction associated with the work that took place to produce the message. If the work was performed within the scope of a transaction, then the BTP context is inserted into the header of the message and the transaction is suspended, which effectively pauses its work for the service until additional messages with a matching context are received.

While none of this is rocket science, it does serve to reiterate that BTP-enabling Web services is a noninvasive procedure, or at least it can be if you choose to adopt a noninvasive strategy. However, at some point every BTP deployment has to interact with existing infrastructure, and it's here that you enter a more intricate phase of development and system integration.

Participants

Participants are the last piece of the puzzle in the BTP architecture (though not quite the last piece of implementation!). You've seen how participants fit into the global BTP architecture, but I haven't yet covered the anatomy of a participant. Participants are the entities that act on behalf of business Web services in matters regarding transactionality, and they're equipped to deal with message exchanges with the transaction manager.

Participants are simply Web services that manage details of distributed transactions on behalf of their associated business services, handling the BTP messages involved in the termination phase of the transaction. While this might sound like hard work, a toolkit will typically simplify matters by offering an interface that your participant will implement in order to become part of the participant stack. The participant stack is shown in Figure 4; the interface that constitutes the API for the stack from the developer's point of view is shown in Listing 1.

Figure 4 shows the conceptual view of a participant (minus the back-end plumbing, which you'll see later). It's a straightforward document exchange–based Web service in which the messaging layer understands BTP messages. It invokes methods on the user-defined participant (which has a known interface) in accordance with the type and content of the messages it receives. Any returns from the participant are shoehorned into BTP messages and sent back through the SOAP infrastructure.

The participant API effectively shields participant developers from having to understand the BTP messages that participants consume, but this shielding isn't entirely "bulletproof," since some understanding of how and when the methods in a participant are called is still required. Listing 1 shows the more important methods that an implementer has to write in order to create a participant. As you might expect, these methods correspond to the messages exchanged between transaction manager and participant (which is itself identified by a unique ID or Uid in the API). As such, if you have an understanding of BTP (which you must have in order to write a decent participant) then the methods are self-explanatory. For everyone else, here's a brief overview:

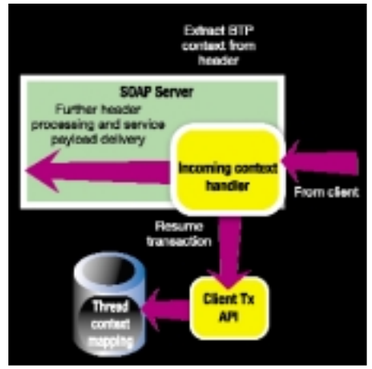


FIGURE 2 **Web service incoming context handler**

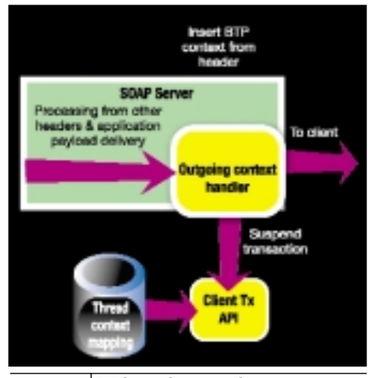


FIGURE 3 **Web service outgoing context handler**

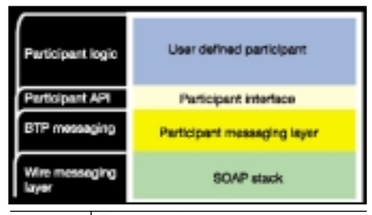


FIGURE 4 **The participant stack**

- ***prepare(…)*:** The prepare method delimits the start of BTP's two-phase confirm protocol. During this method a participant typically checks to see whether it can proceed with a transaction on behalf of the service it's representing, and returns a vote that causes an

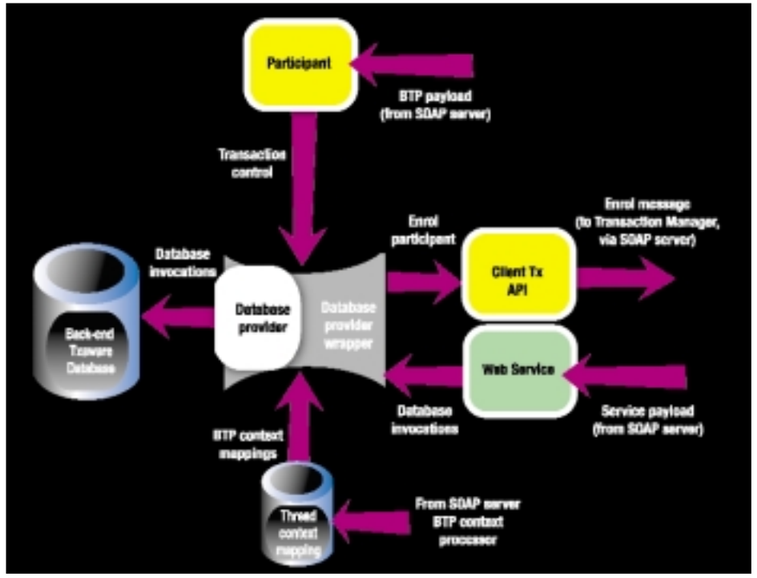


FIGURE 5 | **Integrating services, back-end infrastructure, and participants**

appropriate response to be propagated down the stack and ultimately to the transaction manager. Note that if the participant votes to cancel, it may not receive further messages from the transaction manager.

- ***confirm(…):*** Confirming a participant causes the second phase of the two-phase confirm to occur. At confirm time the participant typically tries to make any changes that the business service has made during its execution durable, for example, by issuing a commit instruction to any underlying data sources.
- ***cancel(…):*** Cancel is the opposite of confirm, whereby a participant will typically try to undo, forget, or otherwise reverse any changes that have been made to system state by the service.
- ***contradiction(…):*** If a service back end finds itself in a situation where it has done the opposite of what it has been asked to do by the transaction manager (e.g., it has canceled when it should have confirmed, or vice versa), and cannot mask the fault, it will send an exception to the transaction manager. The transaction manager will evaluate the situation from a global perspective and may need to inform other participants of the contradiction that has occurred. If this is the case, a participant will learn about contradictions that have occurred when its contradiction method is invoked. At that point a participant typically tries to instigate compensative action; however, to fully recover from a contradictory situation, help from outside the system (even human help!) may be required.

One final intricacy for participants is the sending and receiving of qualifiers. Qualifiers are a neat feature of BTP, derived from the fact that the BTP transaction manager is not as godlike as its equivalents in other transaction management models, but instead accepts the possibility that other parts of the system might justifiably want to help in the decision-making process. Qualifiers support this bilateral exchange of "small print." In essence, each BTP message allows the sender to tag qualifiers that describe such things as, "I will be prepared for the next 10 minutes, and after that I will unilaterally cancel" and "You must be available for at least the next 24 hours to participate in this transaction." In the API, qualifiers are delivered through the Qualifier[] qualifiers parameter (where the transaction manager gets the chance to state its additional terms and conditions) and are returned from the prepare(…) method as part of the vote (where the participant then gets to respond with its own terms and conditions). Qualifiers are a real help when it comes to Web services transactions because in a loosely coupled environment, knowing from the client side that the party you're communicating with will only be around for so long, or being able to specify from the participant side that your party won't hang around while others procrastinate, is invaluable.

Integrating Participants and Services

If context/work association is where the BTP and Web services worlds collide gently, then the integration of participants and services is the real crunch issue. Unlike service-side context handling, sadly, there are no stock answers to the problem of participant-service integration because the strategy adopted will depend on the existing transactional back-end infrastructure that the service itself relies upon. However, you can mitigate this by providing useful tools to the back-end developer in the form of an API that takes care of at least the common tasks. In the same spirit as the client API, two further verbs deal with enlisting and removing participating services from a transaction. Supported by the TransactionManager API, which may be used from the service's back end, they are:

- ***Enroll:*** Enlists a specific participant with the current transaction.
- ***Resign:*** Removes a specific participant from the current transaction.

Using this service-side API and in keeping with the theme of noninvasiveness that's so much in the spirit of BTP, it would be ideal to deploy systems that don't disturb existing (working!) Web services. Fortunately, there are ways and means of doing this.

Figure 5 depicts the back end of a Web service, and is simply the continuation of the diagrams shown in Figures 2 and 3. You can assume that there will be some kind of transactional infrastructure in the back end of most enterprise-class Web services. For the sake of simplicity, here you can assume it's something like a database.

The good news is that even without BTP transactions thrown into the mix, the exposed Web service will still need to talk to its own back-end systems. It's therefore possible to hijack the interactions between the service and the back end to suit your own purposes. A useful strategy in this situation is to wrap the service's database provider within your own provider that supports the same interface, but is also aware of the BTP infrastructure.

In this example, the database provider wrapper has access to BTP context information from the header processing logic embedded in

the service's stack and is aware of the participant service, which performs BTP work on behalf of the business service. Armed with such knowledge, the database wrapper can enroll a participant in the BTP transaction through the enroll operation supported by the API, which causes a BTP Enroll message exchange to occur with the transaction manager. Where there are no upsets during the enrollment of the participant, BTP messages can now be exchanged between the transaction manager and the participant, ensuring that the participant knows exactly what's happening in the BTP transaction at all times.

This knowledge allows the participant to arbitrate between the transactional semantics (if any) of the service's database access and the activity in the BTP transaction. Such arbitration may not be trivial and will certainly require some domain expertise, since the participant implementation will have to reconcile BTP semantics with those of the service's own back-end transaction processing model. For example, a participant implementation might choose to perform a simple mapping of BTP messages to the database, queue, or workflow system equivalents, or the participant might choose to take an optimistic approach and immediately commit all changes to the database and perform a compensating action in the event of a failure. What implementers must remember is that there is no absolute right or wrong, just participant implementations that work well for a given system and those that don't. Time spent analyzing use cases up front will pay dividends in the long run.

The Transaction Manager

Given the transaction manager's importance in the architecture, it might seem strange to mention it at such a late stage and in such little detail, especially since the transaction manager is, after all, the component upon which all other components depend. The paradox is that the transaction manager is simply the least interesting part of the architecture from a developer's point of view. It's a SOAP document exchange–based Web service that implements the BTP abstract state machine and suitable recovery mechanisms such that transactions aren't compromised in the event of failure of the transaction manager. From a development point of view, a BTP transaction manager is simply a black box, deployed somewhere on the network to enable the rest of the infrastructure to work, and only those who have chosen to implement BTP toolkits must worry about its internals.

Bringing It All Together: A Cohesive Example

You've seen the BTP architecture and the SOAP plumbing, and I've touched on the transaction model that BTP supports. Now the many different aspects can be drawn together to form a more powerful example. This example revisits the night out example shown in Figure 1. In part 1 of this article, I showed you some code that interacted with a number of Web services within the context of a BTP atom, thus ensuring a consistent outcome for the services in the transaction. I'll use a similar pattern for this cohesion example, although since cohesions are more powerful than atoms, I'll have to work just a little harder. I'll use approximately the same use case, spicing things up a little by allowing either the theatre or the restaurant service to fail – as long as we get a night out, it's not important what we actually do! Listing 2 shows how to program with cohesions.

The code in Listing 2 follows a pattern similar to the atom example in part 1. You start the transaction and interact with your services via proxies in the normal way. The important difference in the cohesion scenario as compared to the atom example is that the application becomes concerned with the participants that support transaction management on behalf of the services, whereas with atoms the details of any service's participants remain encapsulated by the transaction manager.

There are various ways in which you can obtain the names of the participants that the services enroll into the transaction, but unfortunately the BTP specification doesn't provide any definitive means of obtaining that information (though it does allow participants to be given "friendly names" through the use of a qualifier to help out in such situations). In this example, the services themselves are allowed to report on the names of their participants through their participantID() methods, though any form of a priori knowledge via human or automated means (like UDDI-based discovery) could be feasibly substituted. With time and experience, patterns for this kind of work will no doubt emerge and be embraced by the BTP community.

Once you have the names under which the participants are enrolled in the cohesion, you can then use them to ascertain what decisions the services' participants make when the transaction is terminated. In this example, iterate over the decisions that were returned by the prepare_inferiors(…) call to see whether the taxi and at least one other service are indeed prepared to satisfy the request. If the conditions are met, confirm the transaction with the confirm() method, which confirms all those services' participants that agreed that they could meet your requirements (those that voted to confirm) and cancels any services that couldn't meet your requirements (those that voted to cancel). Conversely, if your overall requirements can't be met, then you can immediately cancel the transaction and the participants will all be instructed to undo the work of their associated Web services.

The power of cohesions therefore arises from the fact that you're at liberty to make choices about who will participate in your transaction right up until the point that you try to confirm it. In fact, you could have used several taxi services in this example to ensure that you got at least one travel option and simply cancelled those that you didn't want before you came to confirm the cohesion.

Similarly, as implementers of the client application, you're at liberty to structure your transactions as you see fit to suit the problem domain. For instance, if you knew that you absolutely had to take a taxi to meet friends at the restaurant, but were not sure whether or not you wanted to go to a show afterwards, you could wrap the taxi and restaurant booking operations within an atom (or indeed wrap several independent taxi and restaurant bookings into several atoms) and enroll that atom in a cohesion along with the participant for the theatre Web service. In this case you're guaranteed to get the taxi and restaurant bookings together (or not at all), while you have some leeway in terms of whether or not you decide to go to the theatre, qualifiers allowing, of course.

Conclusion

Though BTP itself is a sophisticated protocol, from the perspective of an implementer much of its detail is handled by supporting toolkits. As illustrated in the previous article, creating applications that drive BTP transactions is straightforward because of the traditional-looking and intuitive API. In this article, you saw that making Web services transactional is a little trickier, though the toolkits will help to simplify things to a great extent by providing much of the Web service–side infrastructure. The only tough challenge for implementers is the construction of participants, which does require a more thorough understanding of transactional architectures to get right.

This begs the final question: Is BTP a viable technology to roll out with your Web services strategy? The short answer is yes. Since the APIs exposed by BTP implementations are similar to traditional transaction APIs, the learning curve for developers is reasonably gentle. Furthermore, because BTP is among the more mature Web services standards, it has a relatively broad coalition of vendor support. This means that there should be plenty of choices when it comes to picking your toolkit, and a similarly broad range of support from those vendors. All that's left for you to decide is whether the business you conduct over your Web services infrastructure is valuable enough to require transactional support, and when (not if) you will begin your own BTP rollout.

Listing 1

```
public interface Participant
{
    public Vote prepare (Uid id, Qualifier[] qualifiers)
        throws GeneralException, InvalidInferiorException,
            WrongStateException, HeuristicHazardException,
            HeuristicMixedException;


    public void confirm (Uid id, Qualifier[] qualifiers)
        throws GeneralException, InvalidInferiorException,
            WrongStateException, HeuristicHazardException,
            HeuristicMixedException,
InferiorCancelledException;


    public void cancel (Uid id, Qualifier[] qualifiers)
        throws GeneralException, InvalidInferiorException,
        WrongStateException, HeuristicHazardException,
        HeuristicMixedException, InferiorConfirmedException;


    public void contradiction (Uid id, Qualifier[] qualifiers)
        throws GeneralException, InvalidInferiorException,
            WrongStateException;

    // Other methods…
}
```

Listing 2

```
// obtain a UserTransaction object (assume the factory is
// initialized)

UserTransaction userTransaction =
  UserTransactionFactory.getTransaction();

// Obtain references to the web services we are going to use:
RestaurantService restaurant = new RestaurantService();
TaxiService  taxi = new TaxiService();
TheatreService = new TheatreService():

// In a cohesion, we have to make an application-level mapping
//of services to participants (outside of BTP scope). Our
example // services support this.
String restaurantParticipantID = restaurant.getParticipantID();
String theatreParticipantID = theatre.getParticipantID();
String taxiParticipantID = taxi.getParticipantID();

// Start a new transaction, using a Cohesion
userTransaction.begin(com.hp.mw.xts.TxTypes.COHESION);

// Now invoke the business logic
restaurant.bookSeats(3);
theatre.bookSeats(3, 2);
taxi.bookTaxi();

// Prepare (all of) the participants (i.e. with null parameter)
StatusItem[] statusItems =
userTransaction.prepare_inferiors(null);


// Iterate over the statuses, and make sure the taxi's pledged
to honour the booking
boolean restuarantOK = false;
boolean taxiOK = false;
boolean theatreOK = false;
for(int i = 0; i < statusItems.length; i++)
{

if(statusItems[i].inferiorName().equals(restaurantParticipantID
))
    {
        if(statusItems[i].status() == TwoPhaseStatus.PREPARED)
        {
            restaurantOK = true;
        }
    }

    else
if(statusItems[i].inferiorName().equals(taxiParticipantID))
    {
        if(statusItems[i].status() == TwoPhaseStatus.PREPARED)
        {
            taxiOK = true;
        }
    }

    else
if(statusItems[i].inferiorName().equals(theatreParticipantID))
    {
        if(statusItems[i].status() == TwoPhaseStatus.PREPARED)
        {
            theatreOK = true;
        }
    }
}

// If we can't get a taxi, then we have to call the whole //
// event off.
if(!taxiOK)
{
    ut.cancel(null); // Cancel everything
}
else if(restaurantOK || theatreOK)
{
    ut.confirm(); // Confirm whatever is available
}
else
{
    ut.cancel(null); // Can't get anywhere to go, cancel
everything.
}
```

Download the code at sys-con.com/webservices

Written by Mark Little

The Business Transaction Protocol: Transactions for a New Age PART II

Bringing reliability into the Web services world

Use of atomic transactions is a well-known technique for guaranteeing consistency in the presence of failures. The ACID properties of atomic transactions (Atomicity, Consistency, Isolation, Durability) ensure that even in complex business applications consistency of state is preserved.

Transactions are best viewed as "short-lived" entities operating in a closely coupled environment, performing stable state changes to the system; they are less well suited for structuring "long-lived" application functions (e.g., running for hours, days, etc.) and running in a loosely coupled environment like the Web. Long-lived atomic transactions (as typically occur in business-to-business interactions) may reduce the concurrency in the system to an unacceptable level by holding on to resources for a long time; further, if such an atomic transaction rolls back, much valuable work already performed could be undone. As a result, there have been various extended transactions models where strict ACID properties can be relaxed in a controlled manner. Until recently, translating these models into the world of Web services had not been attempted. However, the OASIS Business Transaction Protocol, specified by a collaboration of several companies, has tried to address this issue. In this article, the second in a two-part series, we'll describe how the BTP has attempted to solve these problems.

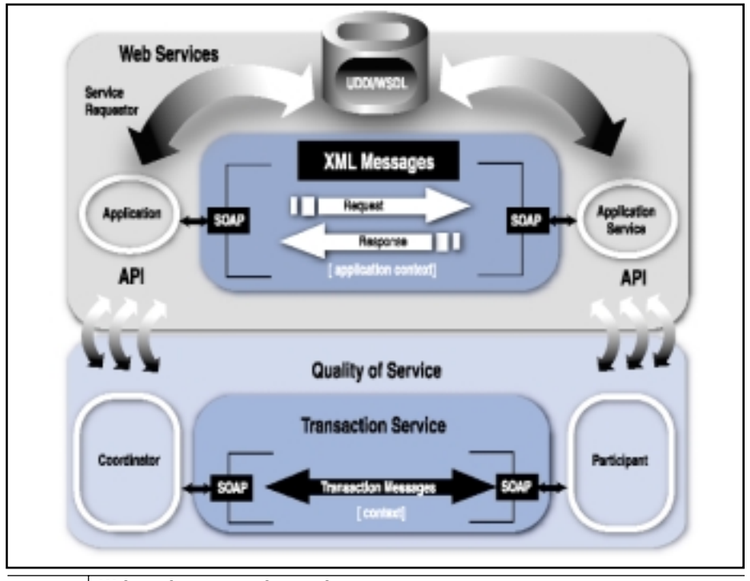


FIGURE 1 | **Web services, transactions and contexts.**

Architecture of the Business Transaction Protocol

A very high-level view of the BTP can be described as follows: Web services do work within the scope of *atoms*, which are created by the initiator of a business transaction; multiple atoms are composed into a business transaction (e.g., arranging a holiday) by a *cohesion composer* such that different atoms may possess different outcomes, as directed by the business logic, e.g., cancel one insurance quote and confirm another. Businesses take part in atomic or cohesive transactions via *participants*, and both cohesions and atoms use *coordination* to ensure that participants see the desired outcome (see Figure 1). This may seem fairly straightforward at first, but as we shall see in the following sections, there's a lot more going on under the covers!

Author Bio

Mark Little is a distinguished engineer/architect within HP Arjuna Labs, where he leads the HP-TS and HP-WST teams. He is one of the primary authors of the OMG Activity Service specification, is on the expert group for JSR 95, and leads the JSR 156 activity on an XML API for Java Transactions. He is HP's representative on the OTS Revision Task Force and the OASIS Business Transactions Protocol specification.
MARK_LITTLE@HP.COM

The XML Context

In order for a transaction to span a distributed number of services/tasks, certain information has to flow between the sites/domains involved in the application. This is commonly referred to as the *context* and typically contains the following information:

- A transaction identifier that guarantees global uniqueness for an individual activity. (Such an identifier can also be thought of as a "correlation" identifier or a value used to indicate that a task is part of the same work activity.)
- The transaction coordinator location or endpoint address, so participants can be enrolled.

The context information is propagated to provide a flow of context information between distributed execution environments, for example using SOAP header information. This may occur transparently to the client and application services. The context is propagated as part of normal message interchange within an application (e.g., as an additional part of the SOAP header).

XML Message Sets and Carrier Bindings

In the Web services world, information is communicated in XML documents, but how those documents are exchanged may be a function of the environment, business relationship, etc. Therefore, although BTP mandates that its own information (context and protocol messages) must be carried in XML payloads, it doesn't specify how these payloads are transmitted; it doesn't mandate a specific carrier protocol.

Obviously, without a carrier protocol, BTP is of very limited use! The technical committee did define a binding to SOAP 1.1 over HTTP 1.1 as part of the BTP 1.0 specification, but the intention has always been that other specific carrier protocol bindings to the BTP XML schema would be provided on an as-needed basis. So if, for example, a group of companies sees merit in defining a binding using pigeons(!), they could so define it and submit it as an appendix on optional bindings to the BTP specification.

As with traditional transaction processing systems, the BTP message set is concerned with messages for driving the protocol and messages containing information for participating within the protocol. The former are typically of interest only to implementers of either BTP or participants, whereas the latter are of interest to service providers and their associated participants.

Typically a BTP message is propagated within the *body* of the SOAP envelope. For example, Listing 1 shows a typical *begin* message.

For application messages that also carry BTP content, the situation is different. In this situation the BTP messages are typically located within the *header* of the SOAP envelope, as can be seen in Listing 2, in which a BTP context is propagated with an application-specific method call.

> "In the Web services world, information is communicated in XML documents"

The Web Service

Whenever a user contacts a Web service, e.g., a taxi booking service, whose work it wishes to be under the control of a transaction, components of the transaction system are responsible for flowing the context to that service. The service can then use this information to enlist a participant with the transaction. The service is responsible for ensuring that concurrent accesses by different applications are managed in a way that guarantees some internal consistency criteria for that service. Note that a Web service may also play the role of a participant.

The Participant

The participant is the entity that does the real transaction work. The Web service (e.g., a theater booking system) contains some business logic for reserving a seat, inquiring about availability, etc., but it will need to be back-ended by something that maintains information in a durable manner. Typically this will be a database, but it could be a file system, NVRAM, etc.

Now, although the service may talk to the back-end database directly, it cannot commit or roll back any changes it (the service) makes, since these are ultimately under the control of the transaction that scoped the work. In order for the transaction to be able to exercise this control, it must have some contact with the back-end resource (the database in our example), and this is accomplished by the participant.

Each participant supports a two-phase termination protocol via the *prepare*, *confirm*, and *cancel* operations. What the participant does when asked to prepare is implementation dependent (e.g., reserve the theater ticket); it then returns an indication of whether or not it succeeded. However, unlike in an atomic transaction, the participant does not have to guarantee that it can remain in this *prepared state*; it may indicate that it can only do so for a specified period of time, and also indicate what action it will take (confirm or undo) if it has not been told how to finish before this period elapses. In addition, no indication of how the prepare is implemented is implied in the protocol, so resource reservation (locking), as happens in an ACID transaction system, need not occur.

The Coordinator

Associated with every transaction type (atom or cohesion) is a *coordinator*, which is responsible for governing the outcome of the transaction. The coordinator may be implemented as a separate service or may be colocated with the user for improved performance. It communicates with enlisted participants to inform them of the desired termination requirements, i.e., whether they should accept (confirm) or reject (cancel) the work done within the scope of the given transaction. For example, whether to purchase the (provisionally reserved) flight tickets for the user or to release them. This communication will be an implementation-specific protocol (e.g., two- or three-phase completion).

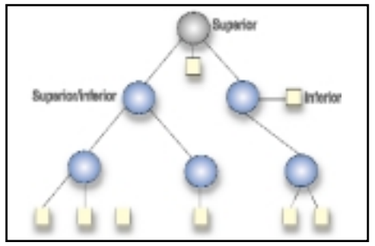


FIGURE 2 | **Transaction relationships**

A transaction manager factory is typically responsible for managing coordinators for many transactions. The initiator of the transaction (e.g., the client) communicates with a transaction manager and asks it to start a new transaction and associate a coordinator with the transaction. Once created, the context can be propagated to Web services in order for them to associate their work with the transaction.

The atom coordinator is typically used to scope work performed on Web services. The cohesion composer is the business logic for gluing together the flow of the application into one or more atoms. Although Web services do work within the scope of a specific atom, it is the composer that ultimately determines which atoms to confirm, and which to undo; as participants are to atoms, so atoms are to cohesion composers (cohesions). The composer may prepare and cancel atoms at arbitrary points during the lifetime of the business transaction, e.g., preparing the flight reservation early in the transaction, and preparing the insurance quote much later after cancelling a prior quote. The main difference between an atom and a cohesion is that whereas all participants enrolled with an atom will either confirm or cancel, the participants enrolled with a cohesion (multiple atoms) may have different outcomes. However, once the composer has arrived at its confirm set (the participants that will confirm), it essentially collapses down to become an atom and guarantees an all-or-nothing effect, i.e., all atoms in the confirm set will either confirm or cancel, with no intermediate effects.

Superiors and Inferiors

Although for simplicity we've talked about services, coordinators, and participants, within BTP all end points are either *Superiors* or *Inferiors* or both. An actor within the coordinating entity's system plays the role of Superior (e.g., the atom coordinator) and an actor within the service plays the role of an Inferior (e.g., the participant). Each Inferior has only one Superior. However, a single Superior may have multiple Inferiors within single or multiple parties. A tree of such relationships may be wide, deep, or both, as shown in Figure 2.

An Inferior is typically associated with some set of application activities. Usually this will be a result of some operation invocations (on a "service application element") from elsewhere (an "initiating application element"). The Inferior is responsible for reporting to the Superior that it is "prepared" for the outcome whether or not the associated operations' provisional effect can be confirmed or cancelled.

A Superior receives reports from its Inferiors as to whether they are prepared to give an outcome. It gathers these reports in order to determine which Inferiors should be canceled and which confirmed. The Superior does this either by itself or with the cooperation of the application element responsible for its creation and control, depending upon whether the transaction is an atom or a cohesion, as we shall see later.

The Initiator

The *initiator* of the atom communicates with an atom/cohesion manager (factory) and asks it to start a new atom. Once created, information about the atom or cohesion (the context) can be propagated to Web services in order for them to associate their work with it. Although work is typically conducted within the scope of an atom, it is entirely possible for services to register participants directly with cohesions.

The Terminator

The *terminator* of the atom or cohesion will typically be the same entity as the initiator, but need not be. For example a long-running stock purchase transaction may be started by the company that requires the stock, and finished by the company that delivers it. Although an atom can be instructed to confirm all participants immediately, it is more typically instructed to prepare them first, and later (hours, days, etc.) to either confirm or cancel them.

Summary

BTP gives builders of transactional Web services the ability to concentrate on the functional aspects of their services (e.g., what it means to book an airline ticket), and to guarantee consensus through the participant interface. Since the participant interface is transparent to its implementation, a provider may use any implementation appropriate to the Web service it acts on behalf of.

Through the cohesion composer, BTP gives the business logic the flexibility to structure interactions with services into multiple (dynamic) consensus groups. The important distinction between BTP and atomic transactions is that multiple such groups exist in BTP, compared to one in atomic transactions, and the cohesion has the capability to drive the two-phase termination protocol explicitly. The fact that atoms may be prepared at any point in the normal flow of business, and later confirmed or undone, gives greater flexibility to the application.

Optimizations

We have described how BTP can be used to conduct typical business-to-business interactions in a reliable manner. In order to do this,

FIGURE 3 **BTP and the Web services stack**

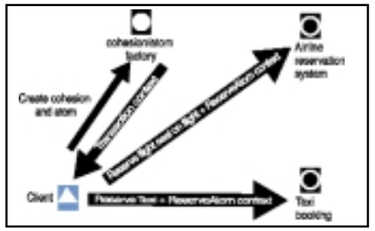


FIGURE 4 **Setting up and using the ReserveAtom context**

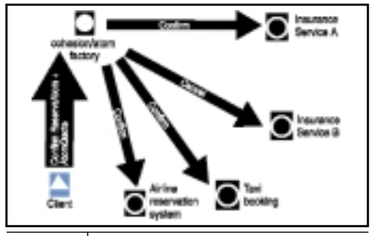


FIGURE 5 **Booking the holiday**

many protocol-specific messages need to be exchanged between actors, and this will have an adverse effect on the time taken to complete a business transaction. This is a necessary side effect of achieving reliability and consensus and is not specific to BTP.

Since BTP is intended for long-running transactions, it may be assumed that performance hasn't been a prime factor in its development. However, this is not the case and, in fact, BTP contains a number of optimizations.

One-Shot

Typically a participant is enlisted with a BTP transaction when a service invocation occurs (e.g., "book flight"). When the service request completes, the response is sent back to the initiator of the request. As described earlier, during transaction termination the coordinator will interact with the participant to ensure completion.

In some circumstances it may be possible to compound many of the above messages into a "one-shot" message. For example, the service invocation may cause a state change to occur that means the participant can prepare immediately after the invocation completes. Rather than having to wait for an explicit coordinator message, BTP allows the enroll request and statement of preparation to be compounded within the service response. The receiver is then responsible for ensuring that this additional information is forwarded to the responsible actors.

Resignation by Participant

In a two-phase commit protocol, in addition to indicating success or failure during the preparation phase, a participant can also return a "read-only" response; this indicates that it doesn't control any work that has been modified during the course of the transaction and therefore doesn't need to be informed of the transaction outcome. In some situations this allows the two-phase protocol to complete quickly, since a second round of messages isn't required.

The equivalent of this in BTP is for a participant to resign from the transaction it was enrolled in. Resignation can occur at any time up to the point at which the participant has prepared. Resignation is used by the participant to indicate that it no longer has an interest in the outcome of the transaction.

Spontaneous Prepare

In some situations, rather than waiting for an instruction from the coordinator to prepare, a participant may be able to spontaneously prepare. For example, a service invocation occurs, moving the service into an idempotent state such that further invocations have no effect on it; in this case, an associated participant may prepare the service immediately, rather than wait for the instruction to do so. In BTP, a participant is allowed to attempt to prepare at any point and inform the coordinator of the result.

Autonomous Decision by Participant

In a traditional two-phase protocol a participant enrolls with a transaction and waits for the termination protocol before it either confirms or cancels. To achieve consensus, it is necessarily a blocking protocol, which means that if a coordinator fails before delivering the final phase messages, prepared participants must remain blocked, holding on to (possibly valuable) resources. Modern transaction-processing systems have augmented the two-phase commit with *heuristics*, which allow such participants to make unilateral decisions about whether they will commit or roll back. Obviously if a participant makes a choice that turns out to be different from that of other participants, nonatomic behavior occurs.

BTP has its equivalent of heuristics, allowing participants to make unilateral decisions as well. However, unlike other transaction implementations, the protocol allows a participant to give the coordinator prior knowledge of what the decision will be and when it will occur. A participant may prepare and present the coordinator with some caveats as to how long it will remain in this state and into what state it will then migrate (e.g., "will remain prepared for 10 days and then will cancel the flight reservation"). This information may then be used by the coordinator to optimize message exchange.

BTP and the Web Services Stack

So where exactly does BTP fit into the evolving Web services architecture? As shown in Figure 3, it is primarily intended as a low-level protocol, hidden from users in much the same way traditional transaction systems are. Typically, a user would see just a demarcation API (e.g., how to start and end an atom); the BTP specification does not define any such API because it is language independent. One possible API that readers should be aware of is that being developed in JSR 156 – Java API for XML Transactions.

So How Would I Use This BTP Thing?

Consider the flight booking example presented earlier. How could we use BTP in order to coordinate this application in a reliable manner? The problem is that we wish to obtain the cheapest insurance quote as we go along, without losing prior quotes until we know that they are no longer the cheapest; at that point we will be able to release those quotes while maintaining others. In a traditional transaction system, all of the work performed within a transaction must either be accepted (committed) or declined (rolled back); the required loosening of atomicity is not supported.

In BTP, however, we can use atoms and cohesions. A cohesion is first created to manage the overall business interactions. The business logic (application, client, etc.) creates an atom (i.e., ReserveAtom) and enrolls it with the cohesion, as shown in Figure 4.

Once the client has obtained the context from the factory, it can invoke the airline and taxi reservation services within the scope of the atom, such that their work is then ultimately controlled by its outcome. When a suitable flight and taxi can be obtained, ReserveAtom is prepared to reserve the bookings for some service-specific time.

Then two new atoms (AtomQuote1 and AtomQuote2) are created and enrolled with the cohesion, before being used to obtain two different quotes from the respective insurance services.

When the quote from the first insurance site is obtained it is obviously not known whether it is the best quote, so the business logic can prepare AtomQuote1 to maintain the quote, while it then communicates with the second insurance site. If that site does not offer a better quote, the application can cancel AtomQuote2 and it now has its final confirmation set of atoms (ReserveAtom and AtomQuote1), which it can confirm (see Figure 5).

Conclusions

ACID transactions have proven invaluable over the years in the construction of enterprise applications. However, they are only really suited to short-duration activities executing on closely coupled applications and environments. When used in a loosely coupled environment, they prove too inflexible and restricting for many applications. The OASIS Business Transactions Protocol has been developed to solve this problem while at the same time maintaining those aspects of the atomic transaction model that have proven useful. At the time of this writing, there is only a single BTP implementation available, from Hewlett-Packard. However, several companies have stated that they are working on their own implementations.

Resources

- *Business Transactions Protocol:* www.oasis-open.org/committees/business-transactions
- *OMG CORBAservices: Common Object Services Specification:* www.omg.org/oma
- *Java Transaction API 1.0.1 (JTA):* http://java.sun.com/products/jta
- *XML Transactioning API for Java (JAXTX):* www.jcp.org/jsr/detail/156.jsp

Listing 1

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP:Envelope
  SOAP:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
  xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Body>
    <btp:begin transaction-type="atom"
      xmlns:btp="urn:oasis:names:tc:BTP:1.0:core" />
  </SOAP:Body>
</SOAP:Envelope>
```

Listing 2

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP:Envelope
  SOAP:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
  xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Header>
    <btp:messages xmlns:btp="urn:oasis:names:tc:BTP:1.0:core">
      <btp:context>
        <btp:superior-address>
          <btp:binding-name>soap-http-1</btp:binding-name>
          <btp:binding-address>
            http://mybusiness.com/btpservice
          </btp:binding-address>
        </btp:superior-address>
        <btp:superior-identifier>
          12fa6de4ea3ec
        </btp:superior-identifier>
        <btp:superior-type>atom</btp:superior-type>
      </btp:context>
    </btp:messages>
  </SOAP:Header>
  <SOAP:Body>
    <ns:myMethod xmlns:ns="http://tempuri.org/">
      <arg1 xsi:type="xsd:int">99</arg1>
      <arg2 xsi:type="xsd:int">101</arg2>
    </ns:myMethod>
  </SOAP:Body>
</SOAP:Envelope>
```

Download the code at sys-con.com/webservices

WSJ News

Altova Releases New XML Tools

(Beverly, MA) – Altova, Inc., has released three new tools to facilitate and advance the adoption of XML technologies.

XMLSPY 5 is the industry-standard XML development environment for designing, editing, and debugging enterprise-class applications. AUTHENTIC 5 is a standards-based, browser-enabled document editor that allows business users to seamlessly capture thoughts and ideas directly in XML format. STYLEVISION 5 is a new XML tool for Web developers that provides powerful conversion utilities for painless migration of traditional HTML Web sites to advanced XML-based sites in full compliance with new standards.

All three products are available for download immediately at www.altova.com/order. Pricing information is available on the Web site.

IBM Unveils WebSphere Portal - Express

(Somers, NY) – IBM has unveiled a new version of its portal software specifically designed for the small- and medium-business market.

WebSphere Portal - Express is particularly simple to deploy and use. It offers full support for portal personalization, campaign management, single sign-on across applications, authentication, authorization, and communication between portlets, and can be run from just one server. Portlets used within WebSphere Portal - Express are reusable across the entire WebSphere Portal family. www.ibm.com

Sonic Software Releases SonicXQ 1.5

(Bedford, MA) – Sonic Software has announced the availability of SonicXQ 1.5. The SonicXQ enterprise service bus is the only enterprise-class integration infrastructure for distributed business environments and computing models.

SonicXQ delivers standards-based integration through Web services and J2EE Connector Architecture technologies. It combines performance and scalability with the service deployment, transformation, orchestration, and administration capabilities required by large-scale integration projects. www.sonicsoftware.com

webMethods Demonstrates Ability to Run Full Business Processes

(Fairfax, VA and San Mateo, CA) – webMethods, Inc., a provider of integration software, and Siebel Systems, provider of multichannel e-business applications software, have announced that webMethods has demonstrated full support of Universal Application Network. The combined integration solution allows global organizations to leverage their IT investments in legacy applications, packaged applications, and integration infrastructure to reduce their overall costs of integration.

Through joint development efforts with Siebel Systems, webMethods is leveraging their current integration server technology to model, generate, import, and export representations of business processes for Universal Application Network. www.webMethods.com, www.siebel.com

Q-Link and iWay Software Partner to Deliver Next Generation BPM

(New York, NY) – Q-Link Technologies, Inc., provider of Business Process Management (BPM) software solutions, and iWay Software, an Information Builders company that accelerates business integration, have announced a technology and OEM partnership agreement in which Q-Link will incorporate iWay's connector technology into its BPM platform solution.

Leveraging iWay's library of more than 200 application and technology adapters and Q-Link's robust process workflow engine and application development environment, the companies plan to offer expanded connectivity capabilities for packaged applications such as SAP, Oracle, Siebel, and more. www.qlinktech.com, www.iwaysoftware.com.

Swingtide Launched to Manage XML Proliferation

(Portsmouth, NH) – A new software company, Swingtide, has been launched to protect, measure, and maximize companies' quality of business amid the spread of XML. Cofounders Jack Serfass and David Sweet were founders of Bowstreet, an XML and Web services infrastructure company, and Preferred Systems Inc., which sold to Computer Associates. Cofounder David Wroe was previously CTO at CNA, a commercial insurer, and chairman and CEO of Agency Management Services, a software company. Swingtide was conceived with the guidance of a group of executives concerned with the emerging complexity of XML as it spreads within their corporations. It is developing software designed to address this problem; their first products are expected to be available later this year. www.swingtide.com

ActiveState Unveils Komodo 2.0 with Web Services Support & GUI Builder

(Vancouver, BC) – ActiveState Corp., a leader in applied open source, has announced Komodo 2.0, a workspace for cross-platform, open-source language programming. Komodo 2.0 affords greater power, flexibility, and automation, including the ActiveState GUI Builder, Visual Package Manager, Source Code Control, Project Manager, Macros, and Web services generation. The only unified open-source programming language IDE, Komodo enables editing, debugging, and testing in a single workspace. www.ActiveState.com.

RSA Security Helps Customers Develop Effective Security Strategy

(Bedford, MA) – RSA Security Inc., has announced the availability of the RSA BSAFE SecurXML-C software development kit for implementing XML digital signatures to ensure the authenticity and integrity of signed data in a Web services environment. RSA BSAFE SecurXML software is designed to help developers build new applications and services faster, reduce development costs, and provide enterprises with a secure foundation for private and legally binding electronic transactions and communications. www.rsasecurity.com

WSJ ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
.NET Developer's Journal	www.sys-con.com/dotnet	888-303-5282	47
Altova	www.altova.com		15
BEA Systems	dev2dev.bea.com/useworkshop		60
IBM	www.ibm.com/websphere/integrate		2
Java Developer's Journal	www.javadevelopersjournal.com	888-303-5282	31, 43
JDJ Store	www.jdjstore.com	800-303-JAVA	29
Parasoft	www.parasoft.com	888-305-0041	13
Richard Hale Shaw Group	www.richardhaleshawgroup.com		27
Sams Publishing	www.samspublishing.com		19
Sitraka	www.sitraka.com/jclassSS/ws	800-663-4723	6
Sitraka	www.sitraka.com/jclass/ws	800-663-4723	59
Sonic Software	www.sonicsoftware.com/websj		9
SpiritSoft	www.spiritsoft.com/climber	508-473-3227	4
Web Services Edge 2003	www.sys-con.com	201-802-3069	53
Web Services Edge World Tour	www.sys-con.com	201-802-3069	35
WebSphere Developer's Journal	www.sys-con.com	888-303-5282	41

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

Beyond the Hype: What to Do with Web Services Today

Paul Holland

Paul Holland is a general partner at Foundation Capital, a leading venture capital firm that focuses on Internet infrastructure, enterprise software, and telecommunications and networking. Paul brings over 18 years of high-technology operating experience to the companies with which he works.

The hype around Web services has been deafening. Equally, there have been plenty of critics awaiting Web services' march behind other over-hyped technologies into the graveyard of "The Next Big Thing That Wasn't." However, when you look at the number of enterprises that have rolled out successful Web services projects to solve real business problems in a relatively short period of time, it is increasingly hard to sound Web services' death knell. From Ford Credit using Web services to give car buyers and dealers real-time access to payment systems and loan application processing to Moody's KMV using Web services to extract and share bond ratings information with its top-tier financial services clients, industry leaders are enjoying tangible business benefits from Web services today.

At Foundation Capital, we believe the time for Web services is now, and we've noticed some key themes common to all successful Web services projects.

The basic building blocks for Web services are stable and mature today. There are accepted standards for description, discovery, and transport. Security and business process management standards are far along the maturation path and currently meet the requirements of the vast majority of the projects being contemplated. With this stable foundation, companies that are moving forward rapidly, like Eastman Chemical and GE Power, are picking projects that match this stage of Web services' evolution. These projects do not have a requirement for reliable messaging, transactions, and complex workflow, whose standards are still at a nascent stage.

What do these projects look like? Generally speaking, Web services projects are most relevant when they involve sharing information that is dynamic, needs to be accessed by a wide variety of constituents, and gains value from being real time rather than static. One project that fits these criteria can be described as B2Bi-lite, in which enterprises streamline interactions with customers and clients outside the firewall based on Web services technology. Examples include exposing inventory levels, credit ratings, and price quotes.

In another successful project, companies are using Web services for cost-effective internal integration between disparate applications and users in a more EAI-lite fashion. Examples include sharing customer information across internal divisions, maintaining integration of packaged applications on varying release schedules, and integrating systems after an acquisition spree.

The most critical aspect of companies' Web services strategies is their post-deployment plan – defined as Web services management and evolution. Even companies rolling out a small number of Web services understand they must have software to help them change and evolve the services environment without disruption. Following are several key features companies need to have in their Web services management solution.

Say a company exposes shipping status as a Web service to its distributors. It must have the ability to monitor and control that service while it is in production. It needs to be able to see the historical and real-time performance of the service, know if the service has gone down, and be able to bring it back online again, all from a centralized view. In addition, it must be able to control and monitor access to the service by its distributors. A good Web services management solution does this.

Managing QoS is probably the most critical aspect of a Web services environment. Consider a leading financial institution that provides integrated reporting to institutional clients by using Web services to collect data, integrate it, and expose it to customers. For this company, the ability to guarantee performance levels is an absolute requirement and, therefore, the service must be able to load balance and failover at the service level, and more importantly, intelligently route requests based on the real-time performance of its services network.

Change is inevitable. Whether from extending application functionality, evolving standards, or changing technologies, the only certainty is that change will occur. Take the financial institution. Say the company wants to roll out a newer version of its credit report Web service and it has 1,000 clients consuming the current service. How will it do so without disrupting these consumers? With a comprehensive Web services management solution, the company will seamlessly upgrade the service without its customers ever knowing a change took place. Additionally, it would have the flexibility to differentiate service and charge more for the newer version. With management software, it can migrate higher paying customers to the new service, while maintaining the older version side-by-side.

The good news is that there are real business cases best suited to be solved by Web services. And there are software solutions in production today for post-deployment Web services management. Foundation Capital's first bet in Web services is Talking Blocks, which has a suite of products to manage Web services. We invested because we believe that Web services management is the critical piece of a full solution set for Web services. Addressing these management imperatives will allow you to leverage Web services to deliver real business benefits and ROI. Today.