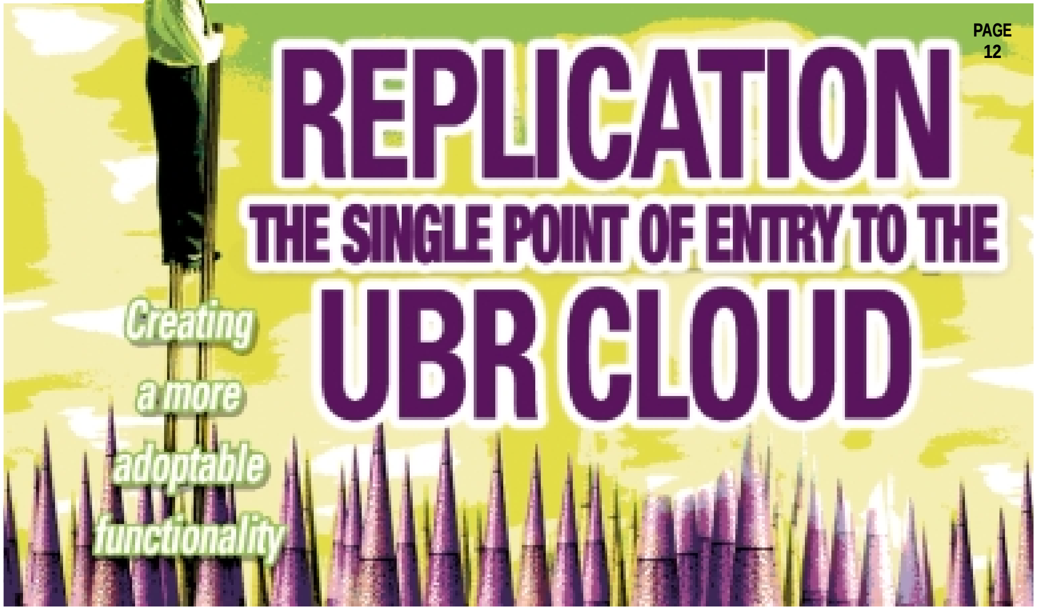
CHOOSING IN THE BATTLE OF TWO IDIOMS

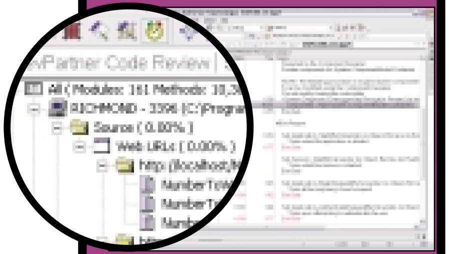
April 2003 Volume 3 Issue 4

WebServices JOURNAL

.NET J2EE XML

WSJ2.COM


PAGE 12
REPLICATION
THE SINGLE POINT OF ENTRY TO THE
UBR CLOUD
Creating a more adoptable functionality



RETAILERS PLEASE DISPLAY UNTIL MAY 31, 2003
$6.99US $7.99CAN

SYS-CON MEDIA

IBM

www.ibm.com/websphere/portalplay

WebServices JOURNAL .NET J2EE XML

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

Get the Message

Written by
Sean Rhody

Author Bio:
Sean Rhody is the editor-in-chief of Web Services Journal *and managing editor of* WebLogic Developer's Journal. *He is a respected industry expert and a consultant with a leading consulting services company.*
SEAN@SYS-CON.COM

Back in the old days, when you needed to communicate with someone distant, you usually had to send a letter. There was no instant response, and there was no way to tell when your message was received. Now we have always-on e-mail, BlackBerrys, and assorted other devices to make what was once a leisurely (or agonizingly slow) process instantaneous, and synchronous.

This issue is about the battle of two idioms – instant, synchronous communication, as championed by the Remote Procedure Call; and asynchronous communication (which may still be instantaneous, but doesn't have to be), represented by the message-based camp.

Web services is about communication, plain and simple. Two (or even more) computers work together to accomplish a task, sometimes directed by human beings, sometimes working based on some program. To work together, the computers exchange information. Web services makes this easy by standardizing the approach to this communication.

But what isn't standardized, at least in theory, is how the communication takes place. That is to say, one approach is the default, but it is not the only allowed, or possible, approach.

By default, a Web service is synchronous. It's simplest to think of it in that sense because an RPC is a very simple concept and the coordination of the interaction is very easy to understand. You simply connect to the service, invoke it, then wait for the response, which you normally would expect instantaneously, or nearly so.

This works well for some things, especially if you expect that the service can provide a quick response. But for other types of applications, particularly ones in which human interaction or workflow might be involved, they are less appropriate. Especially bad is the situation where multiple parties have to be coordinated.

Consider an example where a service might have to arrange for shipping, insurance, bills of lading, even letters of credit. Many parties need to be coordinated here, and not all will be able to respond in a synchronous fashion. In this case, instead of a function call what we really need is to create a conversation.

Of course modeling such a service is inherently more complex. Leaving out the transport mechanisms and callback models for the time being, just drawing a flowchart of such a process might result in pages of lines and boxes, choices and exceptions. And yet, given the nature of the transaction, the complexity is a part of the actual conversation and can't be simplified.

Which brings us back to the reason that Web services doesn't specify how the information needs to be delivered – so that we can define a service, and then define the protocol over which it communicates. We can, for example, use SMTP to move messages over e-mail channels, or use JMS in the Java world to communicate (assuming all the firewalls will cooperate), or use a more generic solution such as TIBCO or MQ Series. Other options are possible too.

But with this flexibility comes a bit more complexity as well. With message-based Web services, we have to devote more thought to what exactly a service, or transaction, is, and how we will know when it is over, and whether it succeeds. These issues are not as important in the RPC world because the transaction is (usually) a single synchronous invocation and as soon as we get the response from the remote system we have the answer. Unfortunately the standards for transaction management, as well as workflow (or business process management), are still evolving, so it will be confusing for a while.

But it's good to have choices. And even now, things that appear synchronous aren't always what they seem. Ask anyone who has my instant messaging address – see if they believe instant messaging is "instant." I can assure you, it isn't.

And the reality is, there is room, and a need for both programming paradigms within Web services. This issue explores those two approaches in depth. Now it's time for me to ignore some instant messages…

Mindreef

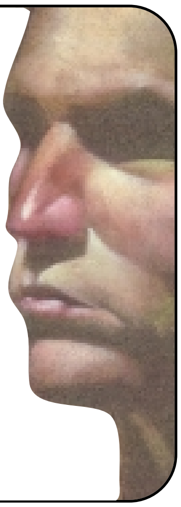
www.mindreef.com/wsj

Mindreef

www.mindreef.com/wsj

Understanding Performance in Web Service Development

Profiling your Web service lets you make intelligent decisions on how to address performance problems



The growth of applications using the .NET platform has generated an increased emphasis on performance measurement and analysis. Distributed applications, while much more flexible and potentially more scalable than monolithic ones, have characteristics that make it more difficult to achieve these very goals.

The problem arises both in the individual components and in their interactions with one another. Individual application components may include computationally expensive code and bottlenecks that don't manifest themselves during unit testing, because the functionality is correct. Once separately developed components are integrated into the full application, performance bottlenecks may result from interactions between them.

These problems are especially true of distributed applications utilizing Web services. In the case of traditional components utilizing COM or CORBA, processing tended to be more synchronous, or at least more tightly coupled, which in turn can result in performance more in line with the overall application. In the case of asynchronous and loosely coupled Web services, there could well be significant differences in their ability to provide the performance and scalability required by the application.

Developers who have worked primarily on stand-alone applications or fat clients may lack direct experience in the performance considerations needed by Web-based distributed applications. In fact, many developers may be surprised at the need to analyze performance and tune individual components including Web services in a .NET application. This doesn't represent a deficiency or limitation of .NET, but rather the reality of the trade-offs required to obtain the flexibility inherent in Web services.

"Web services represent the potential to reuse code components as features across multiple applications simultaneously"

Web services adapt the traditional Web programming model for use from all sorts of applications, not just browser-based ones. These applications are loosely coupled and remarkably interoperable because they can be called from any location that is reachable with a URL or URI, and are not limited by the calling conventions of a specific language.

Microsoft positions ASP.NET as a natural technology for implementing Web services based on the .NET Framework. The ASP.NET Web services infrastructure provides an API for Web services based on mapping SOAP messages to method invocations. ASP.NET Web services support requests from clients using SOAP over HTTP, as well as with HTTP GET or POST operations. ASP.NET Web services automatically generate WSDL files for Web services development efforts. Developers can also use ASP.NET Web services to implement a Web service SOAP listener that waits for service requests and accesses a business facade implemented as a COM component or as a managed code class.

But Web services haven't been proven to stand up to the performance requirements of a wide variety of production uses. That's not to say that they won't or can't, but rather that this type of application model represents a significant unknown in practice.

There are two aspects to measuring and evaluating the performance of an application composed of one or more Web services. The first is the throughput of the Web service itself. This is its ability to accept a request and provide a response in accordance with required performance parameters. On a larger scale, it's also the ability to actually have a transaction throughput that the component was designed to meet. We know that as scalability, although it's difficult to test scalability prior to integrating all application components and functionality.

The second aspect is the ability to evaluate the performance of the application in the aggregate, including the Web service. This includes not only the ability of the Web service to respond, but also how that response is coordinated with the responsiveness of the application as a whole. While the focus may be specifically on the performance of the Web service component, it must be analyzed within the context of the entire application.

Author Bio
Peter Varhol is a product manager at Compuware Corporation and frequent contributor to technology publications.
PETER.VARHOL@COMPUWARE.COM

Parasoft

www.parasoft.com/wsj2

To accomplish this, it's necessary to measure the performance of all of the application components simultaneously, during the same testing run, and correlate those disparate measurements into an integrated view. While the Web service may appear to perform acceptably within its own context, resource or processing issues, synchronization problems, networking, or data throughput, bottlenecks may prevent it from reaching its potential.

Unit Testing During Development

Unit testing a Web service presents the first significant challenge. As a practical matter, it requires an external stimulus to initiate execution, so it isn't simple to call the service as you would a DLL or link it in as you would a library. Fortunately, using the Web services wizard in Visual Studio .NET has the side effect of creating a Web page front end for functional testing purposes, and it works well enough to initiate unit testing also. Otherwise, you would have to write your own call method into the service.

Unit testing should cover both functional and performance testing. Functional testing involves exercising the operations which compose the service, to ensure that they behave as specified. Whether you employ an automated test management system or conduct these unit tests manually, keeping track of code coverage is important to determine what code you've tested and how much you've tested it. Many developers already do this, and while doing so with Web services opens certain challenges, the process is largely familiar.

The goal of performance testing is to identify slow code and potential bottlenecks. For developers with experience in monolithic or tightly coupled applications, this is often simply a matter of noting that the application doesn't perform as expected, isolating the component responsible, and fine-tuning the code. It's not quite like that when building a loosely coupled application with Web services.

However, profiling a .NET Web service is a necessary part of the development process. Since you're not looking at performance or scalability testing of the entire application at this point, you can initiate profiling early in the development cycle.

At a minimum, profiling should collect two types of information – execution time and number of times executed (see Figure 1). The reason for the first is readily apparent – so that you can quickly identify parts of the code that seemingly execute more slowly than others. It's not necessarily indicative of poorly performing code because it may just be performing a computationally intensive operation that can't be improved upon. However, information on the performance of your code justifies a closer look.

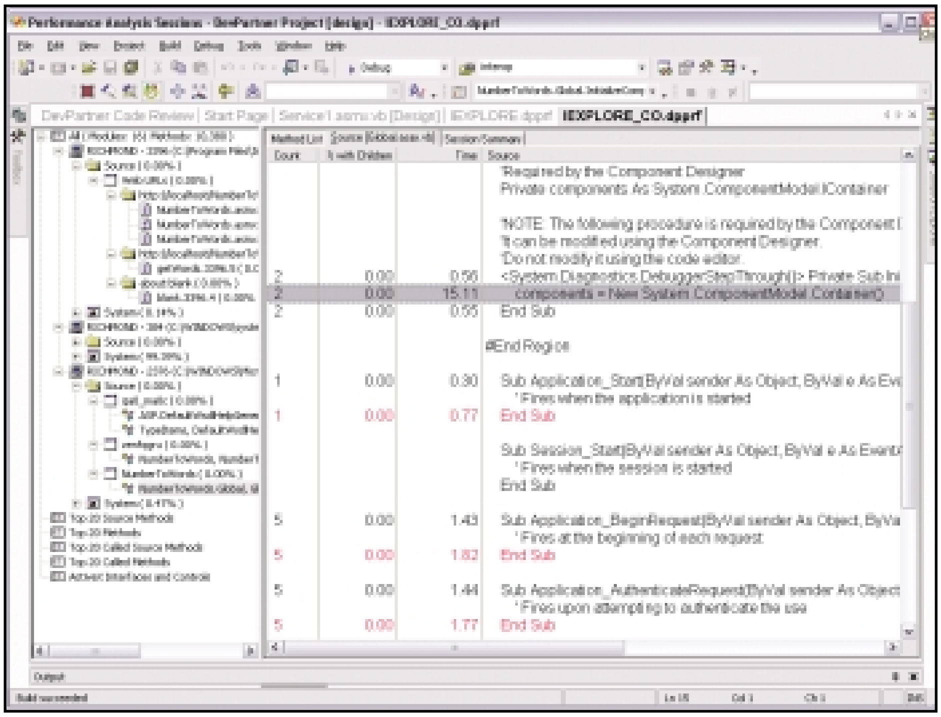
The number of times executed can also be indicative of poorly performing code, but in a different sense. A single line or method may execute in an acceptable amount of time, but may be inefficient in the sense that a single call to that operation does too little work for too much overhead. The trick is to find the optimum amount of data that can be processed most efficiently.

Analyzing Performance and Diagnosing Issues

Profiling a .NET Web service frequently requires looking at both managed and unmanaged code simultaneously. This may be because all but the most trivial calls into native code must undergo a *mode transition*. The mode transition, which is the physical process of moving data between the managed and unmanaged modes of operation, typically requires about two dozen instructions. The second cost is marshalling the data to move across the boundary. Marshalling is computationally expensive, and the more data you move back and forth, the more expensive it becomes.

Alternatively, within the Microsoft model Web services can also be unmanaged. Existing or new COM components can be used to implement the business facade, business logic, and data access layers. Using Windows Server 2003, developers and system administrators can allow existing COM+ applications to be called using XML/SOAP by simply checking a configuration box.

Therefore, you may be making unmanaged calls from your .NET Web service,



Performance Analysis Sessions - DevPartner Project [design] - IEXPLORE_CO.dpprf
File Edit View Project Build Debug Tools Window Help
Debug
DevPartner Code Review | Start Page | Service1.asmx.vb [Design] | IEXPLORE.dpprf | IEXPLORE_CO.dpprf
Method List | Source [Global.asax.vb] | Session Summary

```
Count  % with Children   Time  Source
                               'Required by the Component Designer
                               Private components As System.ComponentModel.IContainer

                               'NOTE: The following procedure is required by the Component
                               'It can be modified using the Component Designer.
                               'Do not modify it using the code editor.
2      0.00              0.56  <System.Diagnostics.DebuggerStepThrough()> Private Sub In
2      0.00             15.11    components = New System.ComponentModel.Container()
2      0.00              0.55  End Sub

                              #End Region

1      0.00              0.30  Sub Application_Start(ByVal sender As Object, ByVal e As Ev
                                 ' Fires when the application is started
1      0.00              0.77  End Sub

                               Sub Session_Start(ByVal sender As Object, ByVal e As Event
                                 ' Fires when the session is started
                               End Sub

5      0.00              1.43  Sub Application_BeginRequest(ByVal sender As Object, ByVa
                                 ' Fires at the beginning of each request
5      0.00              1.82  End Sub

5      0.00              1.44  Sub Application_AuthenticateRequest(ByVal sender As Object
                                 ' Fires upon attempting to authenticate the use
5      0.00              1.77  End Sub
```

FIGURE 1 **Profiling products provides detailed data on execution times and counts on methods and individual lines of code in a Web service.**

Sitraka

www.java.quest.com/jcsc/ws

either because you are using it as a gateway to call COM objects, or are making platform calls, or are using prepackaged native components. Profiling both managed and unmanaged calls together gives you a comprehensive view of the Web service, and most important a view of the performance cost of mode transitions.

The granularity of your profiling should be at least to the method level, and preferably to the individual line of code. At the method level, you can quickly get a view of what operations your code is spending most of its time performing.

In addition to obtaining the execution time and number of calls, you should also be able to analyze your code in several different views of performance, including percent of time in method and percent of time in children. Walking the call stack is an excellent way of determining what resources, .NET services, and OS services your Web service is using, and how expensive those services are (see Figure 2).

Modifying Web Service Performance

Once you have identified slow code, the next step is to address those issues. One popular technique for addressing how data is passed between the application and the Web service is to carefully monitor and optimize the amount of data transmitted in each SOAP call.

In this circumstance, one change you can consider is whether your code should be "chatty" or "chunky." As the names imply, chatty calls are those that occur often and pass little data, while chunky calls occur less frequently but do more work when they do occur. While at first glance it might appear that large calls to the service are more efficient, that's not necessarily true. A chatty interface passing less complex data more often may turn out to be less computationally expensive because its marshalling isn't as complex.

How do you determine whether or not you should be using chatty or chunky calls to your service? There's no easy way that applies to all circumstances; it depends on the amount of data and frequency of calls. The best thing to do is to prototype both the type of interface and the performance profile. By investing a little time early in the development phase, you can ensure that you made the right performance choice and not have to go back and make substantial changes after you have a working application.

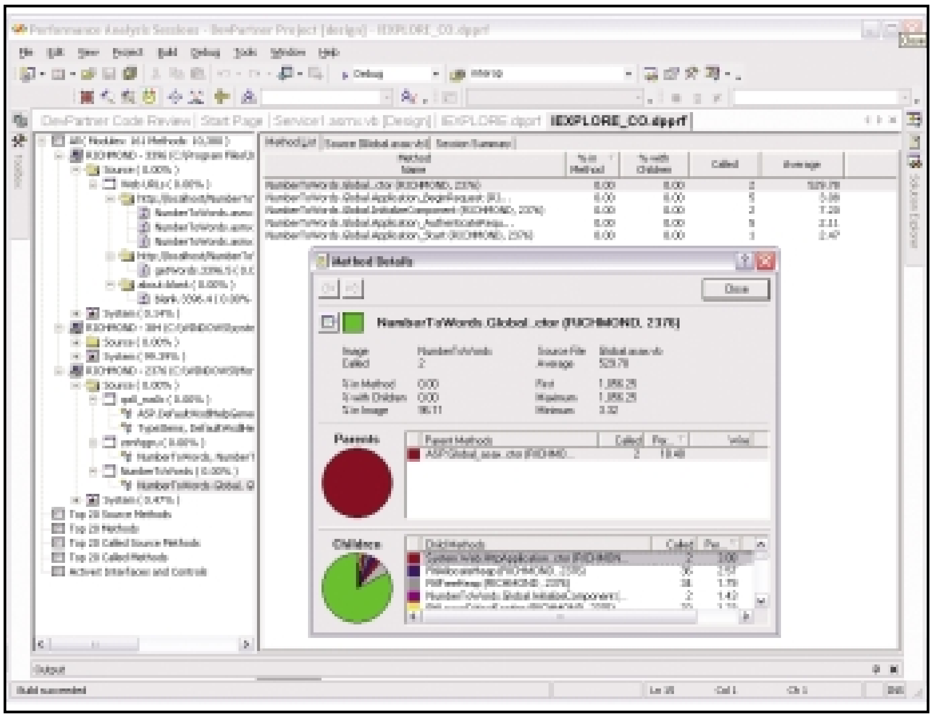
That's not to say that you might not have to go back and make adjustments to your calls once the application is done. For example, you may find that in certain Web services features, chunky calls are more efficient because of the overall volume of calls. But prototyping your data and calls ahead of time gives you a better ability to determine the optimum amount of data to exchange with your Web services in individual transactions.

Another common problem is that certain .NET services can be computationally expensive. At first glance it may appear that you have little control over what the .NET Framework does. However, the framework is rich in features, and there are often multiple ways of obtaining the services you need. Alternatively, the framework calls you are using may be doing more work than you actually need. You won't know any of this unless you have complete profiling information on child behavior from your own methods.

Taking the Uncertainty Out of Web Service Performance

Web services represent the potential to reuse code components as features across multiple applications simultaneously. The loosely coupled model and XML/SOAP communications standard is simple to understand and implement, but the performance implications are not yet well understood. In particular, it's not clear how a Web service used simultaneously by multiple applications will respond to such asynchronous requests.

You're going to be feeling your way in building and testing the performance of Web services. That makes it important to profile these services, both by themselves and within the context of the entire application or applications, to determine where slow code and bottlenecks may reside. And in doing so, you can address those issues by changing the calls to or within the Web service, by modifying expensive database calls, by simplifying complex code paths, or other techniques.


Performance Analysis Sessions - DevPartner Project [design] - IEXPLORE_CO.dpprf
File Edit View Project Build Debug Tools Window Help
Debug
DevPartner Code Review | Start Page | Service1.asmx.vb [Design] | IEXPLORE.dpprf | IEXPLORE_CO.dpprf
Method List | Source Global.asax.vb | Session Summary
Method Details
Close
Parents
Children
Output
Build succeeded

Method Name	% in Method	% with Children	Called	Average
NumberToWords.Global..ctor (RICHMOND, 2376)	0.00	0.00	2	529.78
NumberToWords.Global.Application_BeginRequest (RI...	0.00	0.00	5	5.08
NumberToWords.Global.InitializeComponent (RICHMOND, 2376)	0.00	0.00	2	7.28
NumberToWords.Global.Application_AuthenticateRequ...	0.00	0.00	5	2.11
NumberToWords.Global.Application_Start (RICHMOND, 2376)	0.00	0.00	1	2.47

NumberToWords.Global..ctor (RICHMOND, 2376)

Image	NumberToWords	Source File	Global.asax.vb
Called	2	Average	529.78
% in Method	0.00	First	1,058.25
% with Children	0.00	Maximum	1,058.25
% in Image	98.11	Minimum	3.32

Parent Methods	Called	Pe...	Value
ASP.Global_asax..ctor (RICHMO...	2	18.48	

FIGURE 2 **A key to understanding and improving performance is to have visual information on method calls within your Web service and the ability to walk the call stack to pinpoint performance bottlenecks.**

Altova

www.xmlj.altova.com/authentic5

Written by Arulazi Dhesiaseelan

REPLICATION: THE SINGLE POINT OF ENTRY TO THE UBR CLOUD

Creating a more adoptable functionality

The Universal Description, Discovery, and Integration (UDDI) project became successful in the Web services community after its wide adoption as a Web service discovery protocol. UDDI.org's contribution in defining a standard protocol for services discovery was a major leap forward in service-centric computing. UDDI forms the discovery part of the Web services stack because of its use of existing standards such as XML, HTTP, and SOAP, and its popularity among the developer community. This article looks at the UDDI replication functionality, which is the single point of entry to the UDDI Business Registry (UBR) cloud.

Replication is a process of synchronizing data among the participants (or entities) in the operator cloud. The cloud acts as a single logical entity or entry to the outside world. The goal of replication is to facilitate uniformity and consistency in the data present in the UBR. This can be achieved by the set of replication messages defined in the UDDI Version 2 Replication Specification. Nodes represent operators and are used synonymously in the replication specification. Identified sets of entities form the operator cloud.

AUTHOR BIO:
Arulazi Dhesiaseelan has been designing and building Java-based applications and SDK for more than three years. He was also involved in the API development of UDDI4j project (http://uddi4j.org). Arul works with Hewlett-Packard (India Software Operations), where he is involved in the development of an open service framework for mobile infrastructures.
ARULD@ACM.ORG

This article looks at the importance of replication and its coexistence with the UDDI service. I'll also cover the replication APIs that are implemented by the operators and will discuss the business advantages. I assume you are familiar with XML, SOAP, and UDDI.

UDDI Business Registry Cloud

Figure 1 shows the UBR cloud with the operators replicating each other. Currently, IBM, Microsoft, NTT Communications, and SAP are the public operators of the UDDI registry and form the cloud. The process of adding a node to the cloud is bound to the UDDI Operators council, a governing body within the UDDI.org project. All the public operators should implement the UDDI Specifications as mandated by UDDI.org. Replication as the major functionality is operational with all the operators. With this, Web services clients can query any registry for their businesses, services, tModels, etc., irrespective of their publisher accounts that are bound to a single registry.

"UDDI forms the discovery part of the Web services stack because of its use of existing standards such as XML, HTTP, and SOAP, and its popularity among the developer community"

Replication Business Model

Consider the ACME Company, which specializes in providing consulting related to wealth management for its customers. Assume that ACME registers itself with any of the UBRs (IBM, Microsoft, NTT Communications, or SAP). Suppose that it holds a publisher account with IBM. ACME publishes its business ("ACME Consulting Business") with IBM Business Registry. Now the business, which was published in the UBR, will be visible across the nodes in the UBR through replication. The content of ACME business will also reside in the Microsoft, NTT Communications, and SAP business registries, considering that these entities are involved in replication. These registries will have an entry in their data store that corresponds to "ACME Consulting Business," with IBM as the primary custodian of this business. Likewise, all the services, tModels registered to "ACME business," are replicated across all the UBRs within the cloud. Publisher accounts are not replicated across the registries, only the content or the data in the registry is replicated.

The custodian is the only authoritative person to modify or update the content of the registered business.

Business Search Using UDDI4J

The find_business call of the UDDI4J APIs helps find the replicated business with the operator as IBM, using the Apache Axis as the transport. The snippet of the FindReplicated Business.java is shown in Listing 1 (the listings and sample code for this article can be found online at www.sys-con.com/webservices/sourcec.cfm).

To run this UDDI4j sample, you need to set the classpath which has the following JARs :

```
SET CLASSPATH=
C:\xerces-2_3_0\xmlParserAPIs.jar;
C:\xerces-2_3_0\xercesImpl.jar;
C:\uddi4j-2_0_1\lib\uddi4j.jar;
C:\uddi4j-2_0_1\samples;
C:\ axis-1_1RC1\lib\axis.jar;
C:\ axis-1_1RC1\lib\commons-discovery.jar;
C:\ axis-1_1RC1\lib\commons-logging.jar;
C:\ axis-1_1RC1\lib\saaj.jar;
C:\ axis-1_1RC1\lib\jaxrpc.jar;
```

If you are behind a firewall, run this sample with your proxy details :

```
C:\uddi4j-2_0_1
java -Dhttp.proxyHost=yourProxyHost -
Dhttp.proxyPort=yourProxyPort
FindReplicatedBusiness
```

The output in Listing 2 is the result of a UDDI4J find_business call made to IBM registry. The other UBRs result in similar output with the operator attribute pointing to the respective UBR.

Replication Data Structures

The UDDI Replication Specification defines a set of data structures that are used by the replication APIs.

Update Sequence Number (USN)

The UDDI node participating in the replication process shall assign an increasing number to each of the change records created at that node. This is the originating USN for that particular change record. The originating USN value should be in the increasing order. There can be gaps in the node's originating USN sequence that may be caused by abnormal system failures.

As a result of performing replication, the node has to process all the replicated data, and must assign an additional unique local USN for that particular change record. To avoid the outage of USN values, the replication specification mandates that the nodes should implement a USN with a size exacting to 63 bits. An originating USN of value "0" will be used to represent that no change records have been seen or applied from a node. So, the nodes will skip this USN during replication processing.

Change Records

When a publish call is made to specific datum at a node, the node will create a change record that describes the details of the change. Suppose that when a service is added to a business that exists already, a change record will be generated as a result of this process. The change record will hold the following information:

- ***nodeID:*** Where the change record was initially created
- ***Originating USN:*** Assigned to the change record at its creation by its originating node
- ***Data:*** Conveys the semantics of the change in question.

Change Record Journal

Whenever a node receives change records from other nodes, it should create an entry in the change record journal. The journal stores the XML text of the change records. This helps to verify that the transmitted data has not been altered by the intermediary nodes during the course of replication. The change record journal is maintained in the data store of the UBR.

High Water Mark Vector

Each UDDI node maintains state information such as the originating USN of the most recent changes that have been successfully processed by each node of the registry as a high water mark vector. The high water mark vector has one entry per node with each entry holding the following information:

- ***operatorNodeID:*** The UUID of the node
- ***originatingUSN:*** The originating USN of the most recent change associated with the node that has been successfully consumed

Replication APIs

Replication involves change notification and retrieval of those changes from nodes in the registry. This is done by broadcasting that information from the node in the registry to its peers. The node, which is interested in those changes, will subsequently make a call to retrieve those

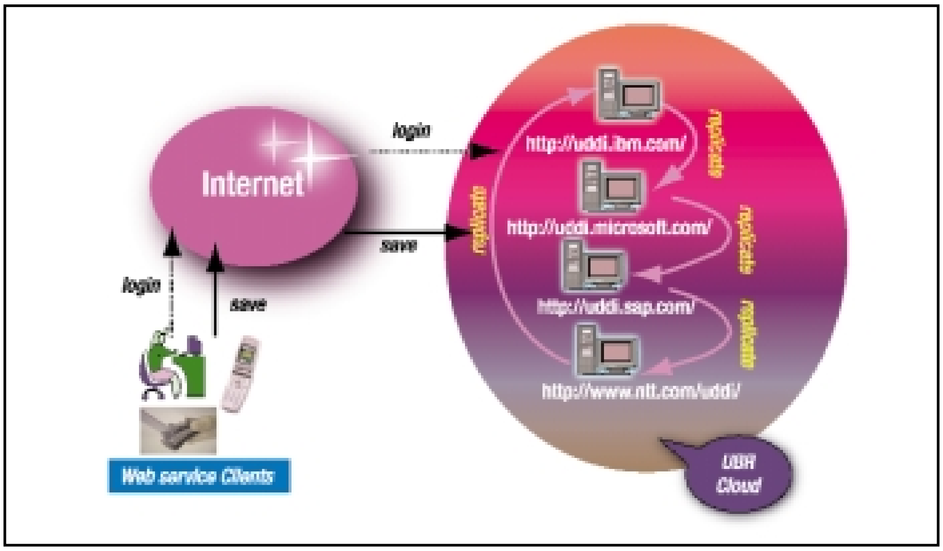


FIGURE 1 | A UBR Cloud with replicating nodes changes. In order to achieve this functionality, UDDI Replication defines the following APIs:

- get_changeRecords
- notify_changeRecordsAvailable
- do_ping
- get_highWaterMarks

get_changeRecords

This UDDI API call is used to initiate the replication of change records from one node to another. The requestingNode is the node that initiates get_changeRecords and will provide information such as chan gesAlreadySeen as part of the high water mark vector. This information is used by the callee to determine the change records needed by the caller.

The get_changeRecords Schema is shown in Listing 3. An example message is shown in Listing 4.

notify_changeRecordsAvailable

Nodes can inform others that they have new change records available for consumption by replication by using this message. The notify_changeRecordsAvailable message is the predecessor to the get_chan geRecords message. The schema for this is in Listing 5 with an example message in Listing 6.

do_ping

This UDDI API call provides the means to verify the connectivity of a node that wishes to start replication.

Schema

```
<element name="do_ping">
 <complexType final="restriction">
  <sequence/>
 </complexType>
</element>
```

Example Message

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xml-
soap.org/soap/envelope/">
    <Body>
      <do_ping xmlns="urn:uddi-
org:repl_v2"/>
    </Body>
</Envelope>
```

get_highWaterMarks

This UDDI API message provides a means to obtain a list of highWaterMark elements containing the highest known USN for all nodes in the replication communication graph.

Schema

```
<element name="get_highWaterMarks">
 <complexType>
  <sequence/>
 </complexType>
</element>
```

Example Message

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xml-
soap.org/soap/envelope/">
    <Body>
      <get_highWaterMarks
xmlns="urn:uddi-org:repl_v2" />
    </Body>
</Envelope>
```

Replication Processing

Replication processing involves the API calls that are made to and from the replicating nodes. This follows a simple life cycle. Consider that nodes A and B (see Figure 2) are participating in a replication scenario. Assume the nodes are configured for replication processing. For instance, Node A initiates the process and makes a "do_ping" call to Node B to check for its availability. Node B makes a similar call to check for Node A's availability. If the "do_ping" call of Node A is successful, then Node A makes a "notify_change Records Available" call to Node B. This call tells Node B that Node A has some changes that are unseen by Node B. In response to this call, Node B makes a "get_changeRecords" call to Node A. Node A sends back all the unseen change-Records to Node B. Node B processes all the change records from Node A and updates its local repository. This completes a single replication cycle, assuming everything goes fine during this process. The replication specification defines a detailed section on the failure scenarios and how to handle them during replication processing. Figure 2 shows the interaction between the replication APIs in case of a two-node scenario.

Replication Configuration

Replication Configuration File

The replication functionality implemented by an operator should be configurable as mandated by the UDDI Replication Specification. This is done through the Replication Configuration File (RCF), which may be located centrally, and can be accessed by the operators. It typically resides in the following URL: https:// www.uddi.org/operator/ReplicationConfiguration.xml and can also be stored within the operator's Web server. In the latter case, each operator has to maintain the same copy of the RCF in order to maintain the consistency of the nodes. UDDI data replication is governed by the set of parameters that form this RCF. This file maintains the necessary information about the operators in the replication process.

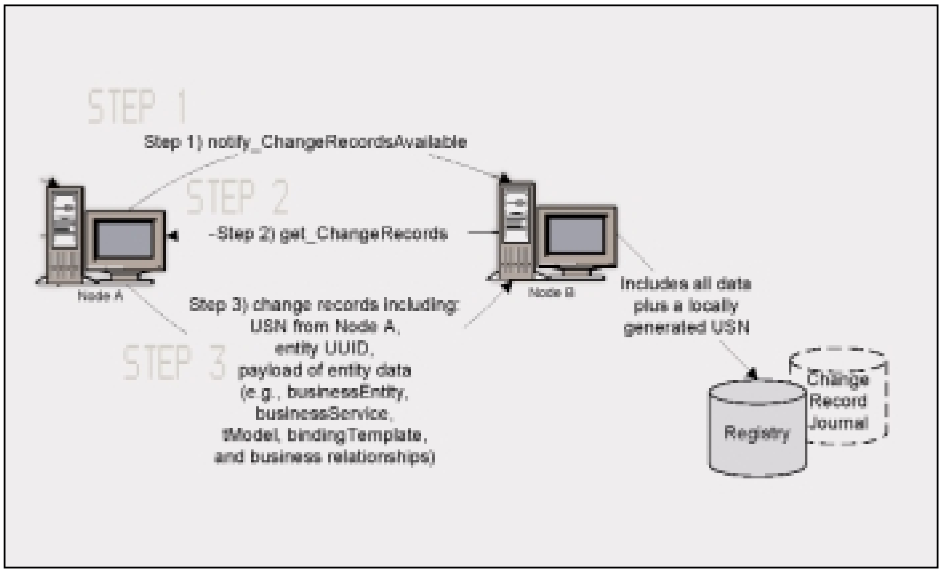


FIGURE 2 | Replication API interaction between two nodes

Global Knowledge

www.globalknowledge.com

“Private registries that are deployed within the enterprise can be promoted to public registries with the help of replication”

The following are the parameters defined in the RCF:

- ***serialNumber:*** Value of this element changes whenever the RCF is updated or changed.
- ***timeOfConfigurationUpdate:*** Gives you the timestamp of the RCF.
- ***councilContact:*** Provides information about the person who maintains or updates the RCF.
- ***maximumTimeToSyncUBR:*** Allows you to specify the maximum amount of time (in hours) that a node in the UBR can sync with all nodes in the UBR. The change made at any single node in the UBR is expected to be visible at all nodes in the UBR within this time limit.
- ***maximumTimeToGetChanges:*** Allows you to specify the maximum amount of time (in hours) that an individual node may wait to request changes. The nodes must perform get_change Records within this time limit.
- ***operator:*** Provides the list of nodes that are part of replication topology.
- ***communicationGraph:*** Provides the communication paths of the nodes and their replication topologies.

Sample Replication Configuration File

The RCF shown in Listing 7 represents a four-node scenario in the communication graph. This RCF holds information about the nodes that take part in the replication process.

Replication Business Advantage

Currently, replication is used in synchronizing the data in the registry among public operators. In the near future, replication as a functionality will be used to synchronize data across geographical locations. Nodes can selectively replicate data based on their requirements. For example, an operator hosting a registry in Japan will be able to replicate only the services that are located in their region based on their language. This filtering helps consumers to avail the services at their doorstep.

Filtering can also be done based on categories, which might be helpful to promote shared businesses. For example, Company A and Company B can host their individual registries, but they can share a specific business segment between them. This business segment has to be replicated between them in order to keep them in sync. Thus, businesses collaborate with each other in a secure manner that benefits their customers.

Private registries that are deployed within the enterprise can be promoted to public registries with the help of replication. Part of the data can be in a private registry and part of it can be in a public registry. For example, you could have a bindingTemplate in a private registry that points to a tModel in public registry. The UDDI v3 Specification details the concept of entity promotion, whereby the test registries can be promoted to production mode retaining their keys.

Summary

As the UBR cloud has become online and operational, the replication functionality of the UDDI registry will make it more adoptable in the Web services community as a standard service discovery protocol. This business-rich feature marks a milestone in the history of UDDI. More businesses should register their meaningful services in the UBR in order to add value to the registry data, which in turn benefits the service consumers.

References

- *UDDI Version 2.0 Replication Specification:* Version 2.03 Specification: http://uddi.org/pubs/Replication-V2.03-Published-20020719.pdf
- *UDDI Version 2.0 XML Replication Schema:* http://uddi.org/schema/uddiv2replication.xsd
- *UDDI Version 2.0 Operator's Specification:* http://uddi.org/pubs/Operators-V2.01-Published-20020719.pdf
- *UDDI4J SDK:* http://uddi4j.org

WebServices JOURNAL .NET J2EE XML

PRESIDENT AND CEO
Fuat A. Kircaali fuat@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

SYS-CON EVENTS
PRESIDENT, SYS-CON EVENTS
Grisha Davida grisha@sys-con.com
CONFERENCE MANAGER
Michael Lynch mike@sys-con.com

CUSTOMER RELATIONS / JDJ STORE
CUSTOMER SERVICE REPRESENTATIVE
Margie Downs margie@sys-con.com
JDJ STORE MANAGER
Rachel McGouran rachel@sys-con.com

SYS-CON.COM
VP, INFORMATION SYSTEMS
Robert Diamond robert@sys-con.com
WEB DESIGNERS
Stephen Kilmurray stephen@sys-con.com
Christopher Croce chris@sys-con.com
ONLINE EDITOR
Lin Goetz lin@sys-con.com

ACCOUNTING
FINANCIAL ANALYST
Joan LaRose joan@sys-con.com
ACCOUNTS RECEIVABLE
Kerri Von Achen kerri@sys-con.com
ACCOUNTS PAYABLE
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: $89.99/YR
ALL OTHER COUNTRIES: $99.99/YR
(U.S. BANKS OR MONEY ORDERS)

SYS-CON MEDIA

Spiritsoft

www.spiritsoft.com/climber

written by Joram Borenstein and Joshua Fox

SEMANTIC DISCOVERY FOR WEB SERVICES

A step toward fulfillment of the vision

The Web services vision of loosely coupled interaction between components, programs, and applications is already beginning to create impressive efficiencies of scale in business integration. The notion of a Web service registry such as UDDI is helping to turn this vision into a reality. Despite this, however, the comprehensive adoption of a truly dynamic discovery of Web services may still be delayed.

AUTHOR BIOS:
Joram Borenstein is a product manager at Unicorn Solutions (www.unicorn.com), working on data semantics technologies and tracking the developing market in semantics. Unicorn leads the EU-funded Corporate Ontology Grid consortium and participates in the W3C Semantic Web Ontology Working group standards activity. Joram's previous experience includes managing the rollout of content management software platforms.
JORAM.BORENSTEIN@UNICORN.COM

Joshua Fox is a software architect at Unicorn Solutions (www.unicorn.com), developing semantic modeling systems for enterprise information unification. He has experience developing large-scale clustered Java systems for Internet collaboration and multimedia delivery, and has published and lectured widely.
JOSHUA.FOX@UNICORN.COM

The Challenge of Dynamic Web Services Discovery and Usage

Currently, the majority of Web services are developed for internal enterprise use. Most existing projects are experimental, while a few early applications already in use provide highly focused functionality for solving specific business and technology problems. For these applications, where all users are within the same enterprise or project team, manual discovery and coding of clients is the most efficient approach.

Web services appear poised to become more widely adopted in coming years, allowing much broader intra- and inter-enterprise integration. Increasingly, developers will require automated systems for service discovery, enabling further Web services interaction with even less human effort. UDDI exists precisely for this reason. However, unless the service's client knows the exact form and meaning of a service's WSDL in advance, the combination of UDDI with WSDL and coarse-grained business descriptions is not enough to allow fully automated service discovery and usage.

Semantic Confusion

When semantics for input and output parameters are weakly defined, increased numbers of Web services in a given functional domain increase the difficulty of accessing each service. Semantic inconsistency can occur in several ways:

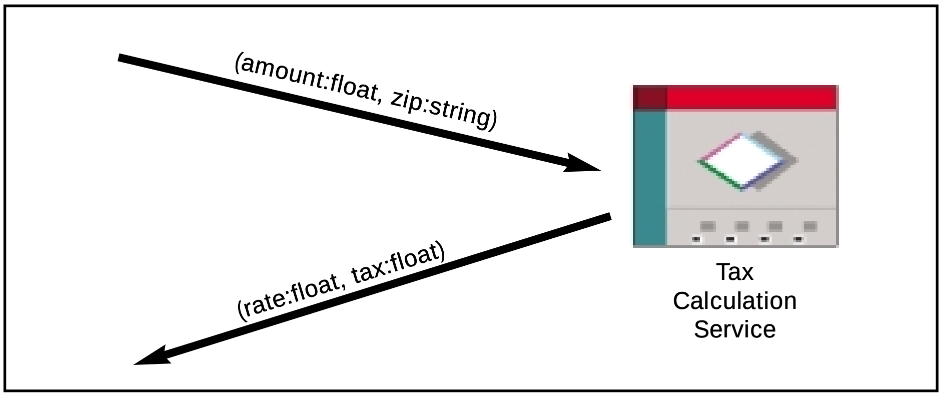


FIGURE 1 A Tax Calculation Web Service

- Poorly defined semantics
- Shared syntax, different semantics
- Shared semantics, different syntax

These problems often combine and result in overlapping and conflicting syntax and semantics.

Poorly Defined Semantics

Today service semantics are often left unstated and are guessed at or interchanged by humans. Without sufficient description, a service is virtually useless to anyone but the original developer.

Let's look at a simple example, a tax calculation service, which has a single, synchronous operation. This operation receives an input message with floating-point amount and string *zip*, and an output message with floating-point *rate* and integer *tax* (see Figure 1).

This is a very simple Web service, yet confusion arises without a formal specification of semantics. For example, what units go into the amount element? It happens to be U.S. dollars. While this might be obvious to the service provider who intended this service for U.S. consumption, potential service clients from other countries need to at least be aware of the expectations of this service.

Shared Syntax, Different Semantics

Unspecified differences in semantics can cause confusion between services. This causes problems when a client is coded to use one service, but tries to use another service with different semantics. As long as the syntax (as represented, for example, by WSDL message definitions) is the same, the call to the service will succeed, but it will return unexpected results.

Two tax calculation services may receive similar inputs and respond with similar outputs, but one calculates income tax while the other calculates sales tax.

The client must be aware of the precise functionality each Web service provides before calling it. Even where WSDL is available, semantics are not expressed precisely in the service interface alone, but rather must be learned by the developer working on the client side.

Shared Semantics, Different Syntax

Likewise, two Web services may have shared semantics (functionality) but different syntax. For instance, two services may calculate sales tax; in the first service, rate is a multiplier – a 5.5% rate would be given as 0.55 – while in the other, it is a percentage, with the same value given as 5.5%. Multiplier and percentage are semantically identical, as long as a factor of 100 can be taken into account in transforming one to the other.

The *zip* element, which allows tax to be calculated based on the legal jurisdiction, is a five-digit U.S. Postal Service ZIP code in one service. Yet a client application may have been coded to work with another service that has the same functionality but instead expects a nine-digit ZIP code. We know that we can derive the short code by truncating the long one, but software does not know this unless it is formally encoded.

Where services have similar functionality but different syntax, it's often quite easy to convert between the two. However, this is only possible if the meaning of each input and output parameter is understood precisely.

“Web services developers can take a significant step toward active Web service integration by overcoming problems of semantic inconsistency”

A Semantic Approach to Service Discovery

Solving these problems requires a semantic approach to the dynamic discovery and interoperability of new Web services, building on lessons learned as part of the implementation of the Semantic Web, a visionary idea of dynamic interoperability of computer systems over the Internet. Tim Berners-Lee, inventor of the World Wide Web, has been promoting his vision for several years (see Resources). The key ingredient of the semantic approach is the formal capture of the Web service's interface by reference to an agreed-upon business-oriented vocabulary (semantic model), described in more detail later.

Although most writings on the Semantic Web do not explicitly mention SOAP, WSDL, and UDDI, the vision of the Semantic Web is best implemented with Web services. In this article, besides presenting a practical solution to a practical problem faced by Web service developers today, we hope to show how Web services fit into the vision of the Semantic Web.

The architecture for Web service discovery through UDDI, or other Web services registries such as ebXML, breaks down interaction stages into distinct roles, including Service Provider, Registry, and Client. To add semantics to these interactions, no change is required for the UDDI Registry, so roles in the semantic approach can be broken down into Provider and Client. Of the two, the Provider has the greatest burden, providing more detailed information about the meaning of the service. The developer on the client side must also change his or her workflow to use the semantic approach. No changes are needed to the discovery process are needed, but with the use of a semantic description, the client can discover the service semantically, and then apply transformations to adapt the interface of the service to the interface expected by using existing Client software.

Despite these changes, the semantic approach simply requires the formalization of existing development steps; all the semantic steps must in any case be done, albeit informally, in spreadsheets or word-processor documents. For a service to be used, the Provider must somehow tell the Client what the service “means,” and the Client must understand what it means and adapt the expectations of the client software to the actual Service interface.

The Provider's Role: Semantic Registration

The Service Provider Development Process

- Use an ontology to model the real-world concepts related to the service's functionality.
- Create a WSDL document for the Web service.
- Map elements of the WSDL messages to semantic concepts, saving mappings in RDF.

- Register the model, WSDL, and mappings to the UDDI Registry.

The role of the Service Provider is to formally represent the service's semantics in a machine-readable way in a UDDI Registry.

Model Real-World Concepts

To enable semantic queries, a Service Provider starts by modeling the semantics of the real-world concepts associated with the service. Such a model is based on the principles of ontology, the science of meaning. A semantic model based on ontology includes:

- ***Classes:*** Sets of real-life entities with some characteristics in common, and the generalization/specialization (inheritance) relationships between them
- ***Properties:*** Relate these classes to each other
- ***Business Rules:*** Indicate constraints on and relationships between properties

Part of the model for our example is depicted in Figure 2, while a fuller version is available as Listing 1 (for space reasons the code for Listing 1 is online at www.sys-con.com/webservices/ sourcec.cfm).

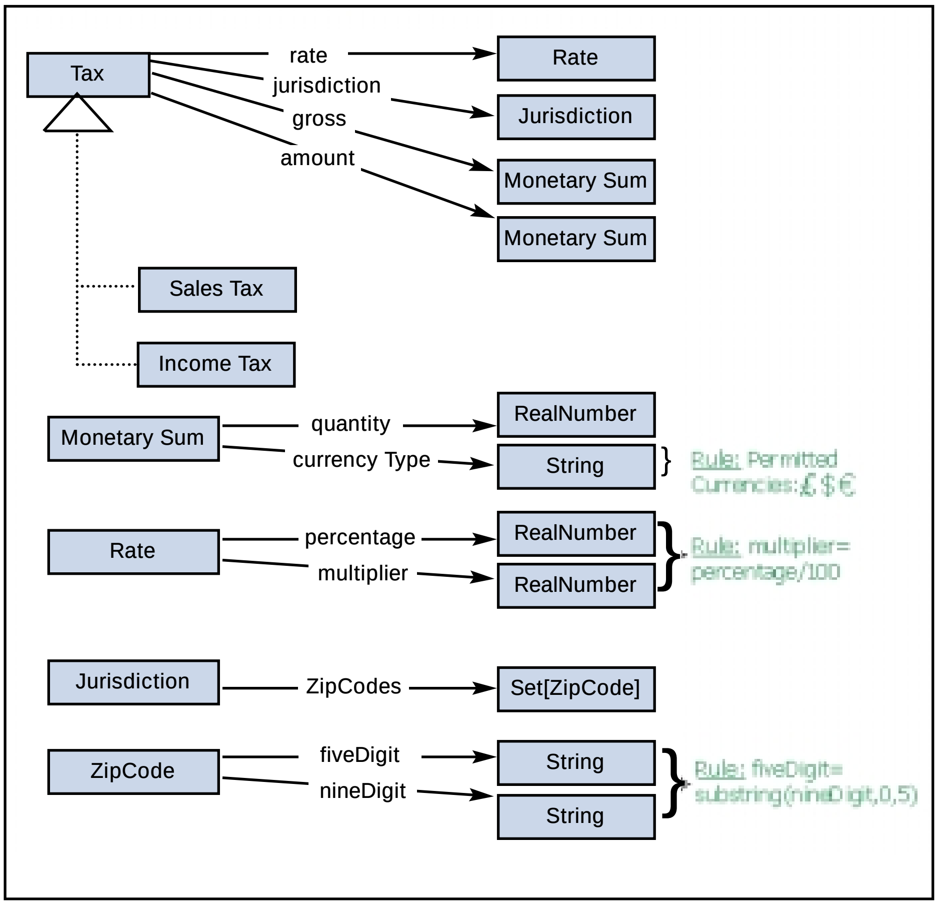


FIGURE 2 **Semantic model**

"Roles in the semantic approach can be broken down into Provider and Client"

OWL (Web Ontology Language) is a markup language semantics that enables ontology sharing via the Web and is the W3C's XML-based working draft for a standard (see Listing 1). It is an updated version of DAML+OIL (DARPA Agent Markup Language + Ontology Inference Language) and continues much of the work done previously on RDF (Resource Description Framework). The OWL representation can be registered easily in a standard UDDI Registry as a tModel, as we will explain later.

The semantic model represents the meaning of a service's functionality. Achieving consensus in ontology can be difficult. But even before standards emerge for the ontologies of specific vertical business areas, the Service Provider can register a semantic model, which the Client can then inspect and use for look-up. Where heterogeneous semantic models must be reconciled, they can be integrated by merging them into a larger model that includes synonymous classes and properties.

With a formal semantic model in place, it doesn't matter if the service works with dollars or euros, percentages or multipliers, as long as the service's functionality is encoded in the ontology. With formalisms in place, such syntactic differences are easily overcome through XSLT transformations that convert from one syntax to another.

The ontological model does not yet represent the service itself, since specific input and output parameters are not encoded. However, it does represent the critical concepts needed to understand the service's functionality.

Create a WSDL Document for the Web Services

Next, the Provider creates the WSDL, which defines the Service's syntax by specifying the structure of the input and output messages, along with aspects of the service's runtime bindings. Creating a WSDL document and registering it to a UDDI Registry is a best practice even if the semantic approach is not used. A WSDL document for this service is shown in Listing 2.

Map Elements from the WSDL Messages to Semantic Concepts

The next stage is to express the meaning of the WSDL by mapping the operations, along with the schemas of the WSDL input and output messages or signatures, to the semantic model.

In some cases, we map a schema element into an indirect property, which is a series of several properties, to express semantic concepts precisely.

Kenetiks

www.kenetiks.com

In our example, we map the schema element rate in the output structure to the rate property of class Tax, linked to the multiplier property of class Rate. In this way, we have encoded the precise meaning of this output parameter, stating that the rate is expressed as a multiplier rather than as a percentage. A Client looking for a percentage can later transform multiplier to percentage by using the business rule formally encoded in the semantic model: multiplier = percentage/100.

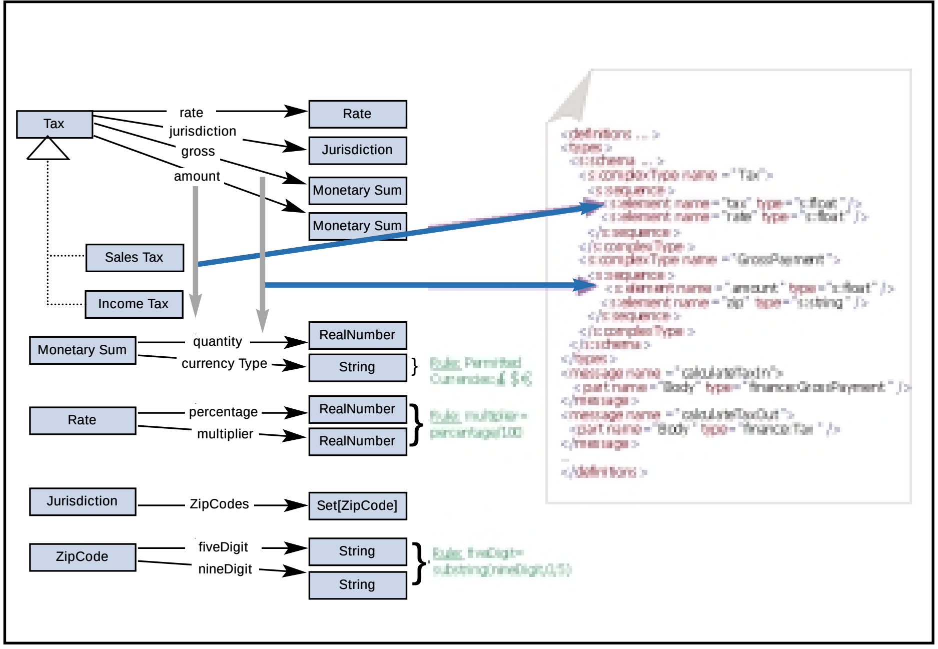
Likewise, we map the zip schema element to the jurisdiction property of class Tax linked to the zipCodes property of class Jurisdiction, linked yet again with fiveDigit property of class ZipCode. In this way, we state that the tax is to be calculated in a given jurisdiction (city or state), that the jurisdiction is characterized by its ZIP codes, and that the ZIP code is expressed in its five-digit form.

Figure 3 shows a subset of these mappings graphically, while Table 1 lists the mappings in full.

Mappings can be expressed in an XML document formatted in the RDF. RDF is built of triplets expressing a relationship of the form subject-predicate-object. For example, we would express a mapping as the following triplet: 1) "element rate", 2) "maps to", and 3) "property rate of class Tax".

Register the Model, WSDL, and Mappings to the UDDI Registry

The last stage is to register these elements to a UDDI Registry. The concepts in the WSDL, the semantic model, and the mappings between them fit quite naturally into UDDI as tModels. In fact, it is already considered a best practice to store WSDL in this way; the other artifacts are stored in the same way (see Figure 4).



```
<definitions ... >
<types>
  <s:schema ... >
    <s:complexType name ="Tax">
      <s:sequence>
        <s:element name="tax" type="s:float"/>
        <s:element name="rate" type="s:float"/>
      </s:sequence>
    </s:complexType>
    <s:complexType name ="GrossPayment">
      <s:sequence>
        <s:element name ="amount" type="s:float"/>
        <s:element name="zip" type="s:string"/>
      </s:sequence>
    </s:complexType>
  </s:schema>
</types>
<message name ="calculateTaxIn">
  <part name="Body" type="finance:GrossPayment"/>
</message>
<message name ="calculateTaxOut">
  <part name="Body" type="finance:Tax"/>
</message>
...
</definitions>
```

FIGURE 3 **Some of the semantic mappings**

The semantic approach to UDDI requires registering more artifacts to the registry than the nonsemantic approach. Fortunately, however, the UDDI object model is flexible enough to allow this sort of usage since the UDDI API has been designed with batch-processing capabilities, allowing the registering or querying of multiple tModels with a single remote call.

The Client's Role: Semantic Discovery

- Identify the required functionality.
- Query UDDI, locate service, and retrieve all semantic and interface descriptions.
- Create transformations to use the service from the Client.
- Access the Web service using the WSDL document.

The Web service is now ready for semantic discovery. The Client needs to discover a service that meets the required business needs and then calls this service.

Identify the Required Functionality

To identify the required service functionality, the Client first discovers the agreed-upon semantic model using UDDI and loads it over standard HTTP. The techniques to do so resemble the use of UDDI for finding any other resource. The Client simply locates the OWL document representing the semantic model by finding the appropriate tModel based on industry sector and general taxonomies.

Even when not standardized, the semantic model is general enough to cover a wide variety of services for a given industry vertical or horizontal sector. For example, the model depicted above (see Figure 2 and Listing 1) would be part of a much larger industry-standard model for financial services. Where multiple "standard" models exist, they can easily be linked together through ontological properties and inheritance in order to form a larger, more inclusive model.

In the non-semantic approach, the industry definitions and taxonomies jointly express a kind of semantics. But the Client must still know exactly what service is needed and exactly what input and output messages to look for in the registry. In the semantic approach, the Client first accesses the general service category through tax-

TABLE 1: **Semantic Mappings in Full**

Ontological Indirect Property	Schema Element
Tax.gross → MonetaryAmount.quantity GrossPayment	*amount* in complex type
Tax.jurisdiction → Jurisdiction.zipCodes → ZipCode.fiveDigit GrossPayment	*zip* in complex type
Tax.amount → MonetarySum.quantity	*tax* in complex type *Tax*
Tax.rate → Rate.multiplier	*rate* in complex type *Tax*

onomies; then, by referencing the functionality of the service in meaningful semantic classes such as *SalesTax*, *ZipCode*, and *Jurisdiction*, the Client finds the service that approximates its precise needs.

Query UDDI, Locate Service, and Retrieve all Semantic and Interface Descriptions

Having identified the relevant ontological concepts in the semantic model, the Client now navigates the mappings that link the model to the required WSDL files. The Client discovers a WSDL file with a function whose input and output schemas are mapped to the input and output concepts.

In our example, a suitable Web service is any WSDL with the correct semantics:

- The input-message schema includes elements expressing the ontological properties amount and zip.
- The output-message schema includes elements expressing the ontological properties tax and rate.

The Client identifies these semantic values and uses the mappings to find identifiers for the services that provide the functionality. The Client can then directly discover these services through UDDI.

Create Transformations to Use the Service from the Client

The Client may have been coded to pass input and expect output with a certain syntax, while the service expects input and returns output with another syntax. For example, the Client may pass a ZIP code as a nine-digit string, while the service expects a five-digit string; likewise, the service may return a tax rate as a multiplier such as 0.055, while the Client expects a percentage such as 5.5.

This problem can be resolved through semantics. In a simpler but manual approach, the developer can code the transformation from one to the other using XSLT. With semantics in hand, the meaning of the parameters is clear and such transformation code is easy to develop. In a more advanced and dynamic approach, semantically based software can automatically generate XSLT to convert from the XML of one schema to the XML of another schema, based on the shared semantics of the schemas. This is suitable for WSDL operations whose parameters are specified as "literal." A third approach combines semantics with the flexibility of DII (Dynamic Invocation Interface): the client adapts its input messages to the needs of the service's interface at runtime. This is suitable to parameters specified as "encoded."

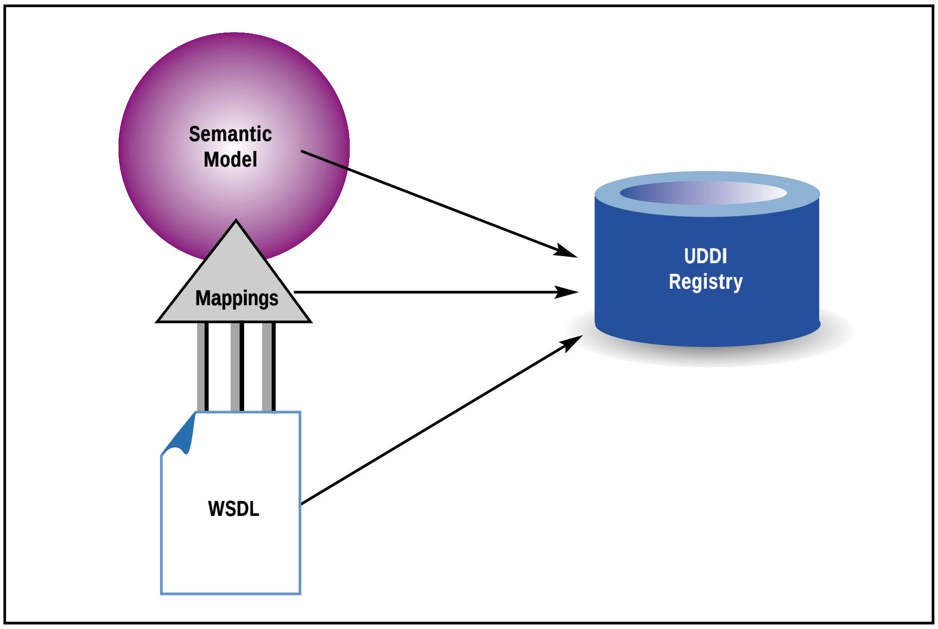


FIGURE 4 Registering the semantic model, WSDL, and mappings to the UDDI Registry

"Without a semantic infrastructure to form the backbone of the UDDI Registry, semantic inconsistency will prevent the large-scale adoption of active Web Services adoption"

Access the Web Service

With a WSDL document available to fully characterize the Web service and transformations in place to adapt between the Client and the Server, the Client can now call on the Web service directly, just as with any WSDL.

Conclusion

In the context of UDDI, WSDL alone cannot deliver the vision of loosely coupled, dynamically interoperating Web services, and so cannot ensure that the current growth in Web services adoption will continue. Without a semantic infrastructure to form the backbone of the UDDI Registry, semantic inconsistency will prevent the large-scale adoption of active Web services adoption. The ongoing decentralization of Web services will make this problem increasingly acute in the coming few years. Formalization of the semantics in Web services is

WSDL Message Syntax

WSDL allows SOAP input and output to be specified in two different ways: as "literal" values or "encoded" as function parameters. In the former approach, XML documents are passed; the latter approach resembles function calls as seen in programming languages. Either the "literal" or "encoded" approach can be used with the technique described here. Examples are given with the "literal" approach.

a step toward automated UDDI discovery and the fulfillment of the Semantic Web vision.

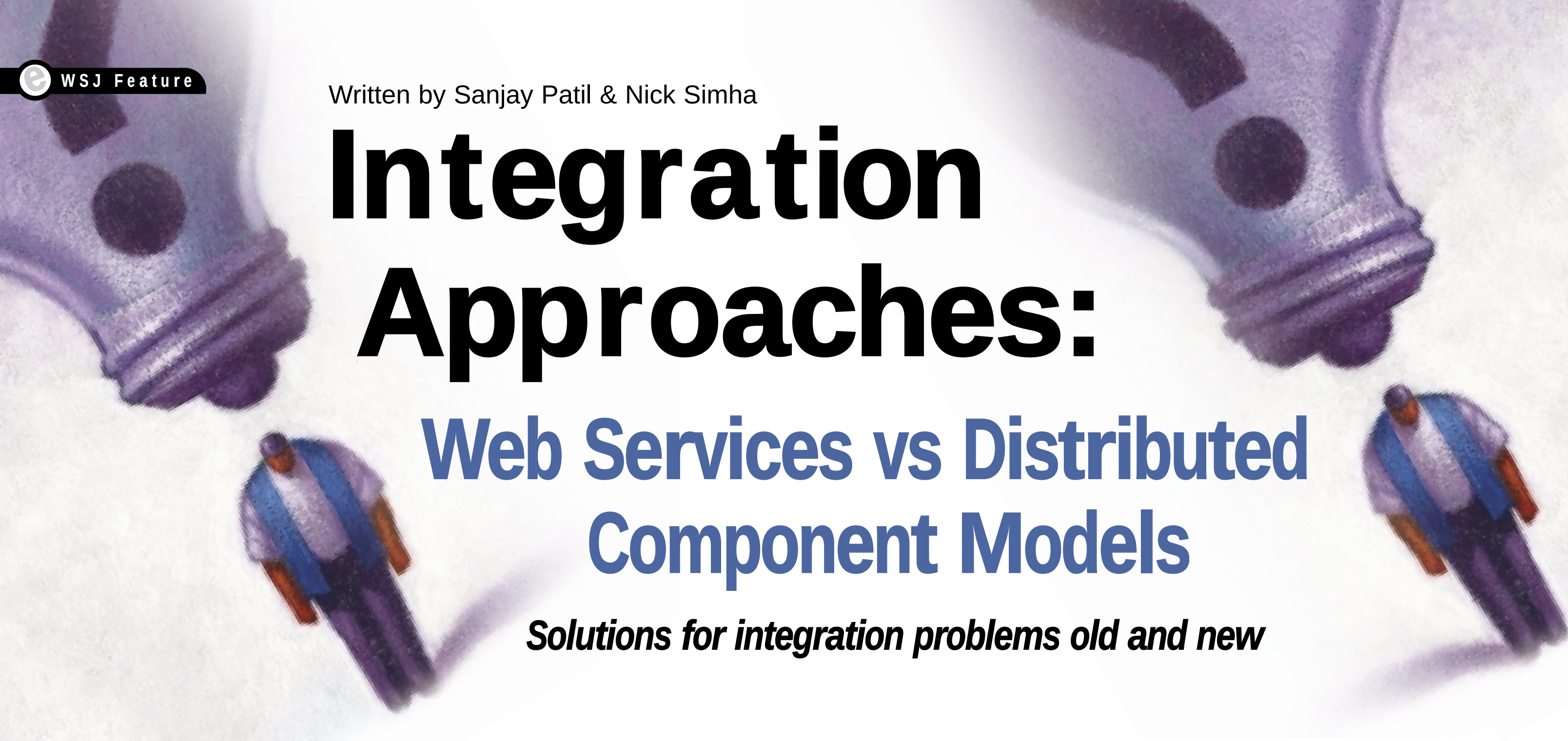
Resources

- *UDDI for Python (UDDI4Py)*: www.alpha-works.ibm.com/tech/uddi4py
- *Microsoft UDDI Software Development Kit*: http://uddi.microsoft.com/developer/default.aspx
- Java API for XML Registries (JAXR): http://java.sun.com/xml/jaxr/
- *UDDI Homepage*: www.uddi.org
- *UDDI4J (Open Source UDDI Client API in Java)*: www.uddi4j.org
- *"Using WSDL in a UDDI Registry*: www.oasis-open.org/committees/uddi-spec/doc/bp/uddi-spec-tc-bp-using-wsdl-v108-20021110.htm
- *Resource Description Framework*: www.w3.org/RDF
- *"Semantic Web Activity (W3C)"*: www.w3.org/2001/sw/
- *Web Ontology Language (OWL) Reference*: www.w3.org/TR/owl-ref
- *Berners-Lee, Tim; Hendler, James . and Lassila, Ora. "The Semantic Web."* www.sciam.com/2001/0501issue/0501berners-lee.html
- Grove, Andy. "Understanding UDDI tModels and Taxonomies," ***Web Services Journal*** (Vol. 1, issue 2; November 2001).
- Januszewski, Karsten. "UDDI and WSDL: Natural Companions," ***Web services Journal*** (Vol. 1, issue 3; December 2001).
- Ogbuji, Uche. "Supercharging WSDL with RDF." www-106.ibm.com/developerworks/ library/ws-rdf/?dwzone=ws

Listing 2: **WSDL file**

```
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:finance="http://www.financialexample.com/TaxService/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://www.financialexample.com/TaxService/
">
 <types>
  <s:schema
targetNamespace="http://www.financialexample.com/TaxService/
" elementFormDefault="qualified">
   <s:complexType name="Tax">
    <s:sequence>
     <s:element name="tax" type="s:float"/>
     <s:element name="rate" type="s:float"/>
    </s:sequence>
   </s:complexType>
   <s:complexType name="GrossPayment">
    <s:sequence>
     <s:element name="amount" type="s:float"/>
     <s:element name="zip" type="s:string"/>
    </s:sequence>
   </s:complexType>
  </s:schema>
 </types>
 <message name="calculateTaxIn">
  <part name="Body" type="finance:GrossPayment"/>
 </message>
 <message name="calculateTaxOut">
  <part name="Body" type="finance:Tax"/>
 </message>
 <portType name="TaxServiceSoap">
  <operation name="calculateTax">
   <input message="finance:calculateTaxIn"/>
   <output message="finance:calculateTaxOut"/>
  </operation>
 </portType>
 <binding name="TaxServiceSoap"
type="finance:TaxServiceSoap">
  <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="calculateTax">
   <soap:operation
soapAction="http://www.financialexample.com/TaxService/calcu
lateTax" style="document"/>
   <input>
    <soap:body use="literal"/>
   </input>
   <output>
    <soap:body use="literal"/>
   </output>
  </operation>
 </binding>
 <service name="TaxService">
  <documentation>A sample Time service</documentation>
  <port name="TaxServiceSoap"
binding="finance:TaxServiceSoap">
   <soap:address
location="http://www.financialexample.com/TaxService/TaxServ
ice.jsp"/>
  </port>
 </service>
</definitions>
```

Download the code at
sys-con.com/webservices



Written by Sanjay Patil & Nick Simha

Integration Approaches: Web Services vs Distributed Component Models

Solutions for integration problems old and new

This article, the first of two parts, will compare and contrast Web services with other distributed computing component technologies such as CORBA, J2EE, and DCOM. We look at these approaches in the context of their respective capabilities in support of integration solutions and application architecture domains (e.g., loosely coupled versus tightly coupled applications).

These technologies are complementary, but the most important consideration for choosing a particular technology is its suitability for a particular problem/solution domain.

At a high level, an enterprise application environment consists of *domain-specific applications*, as well as certain *integration applications* that involve weaving together parts of the domain-specific applications. For domain-specific applications, it is efficient and feasible to make use of technologies such as J2EE and CORBA, as they provide a solid application development and runtime platform.

While it is technically possible to use J2EE, CORBA, and similar technologies for integration applications, products in the enterprise application integration (EAI) category (which includes, for example, message-oriented middleware products) are a better fit. Web services technologies are best suited for the EAI problem domain and are complementary to the problem domain of J2EE, CORBA, and other distributed component technologies.

In this article we describe the characteristics of the enterprise integration environment, the attributes of an ideal integration solution, and the architectural requirements entailed by these considerations. We look at how Web services can provide solutions for long-standing integration problems, as well as for new classes of integration problems that are not explicitly addressed by component models.

Finally, we'll look at how distributed technologies such as CORBA are better suited for domain-specific application development, and how they complement Web services technologies.

AUTHOR BIO:

Sanjay Patil is a distinguished engineer within the Web Services Integration Platform group at IONA Technologies. He is actively involved with many e-business standards; his areas of interest are distributed infrastructure for B2B and enterprise integration, Web services, and Internet technologies.
SANJAY.PATIL@IONA.COM

Nick Simha is a sales engineer working with IONA's customers to help architect e-Business solutions. His areas of interest are application integration, Web services, supply chain management, and security.
NSIMHA@IONA.COM

Enterprise Integration Environment

This section describes some of the characteristics of the integration environment for the enterprise, including the technical landscape and specific problem areas typically addressed by integration projects. Most of these problems are commonly applicable to internal integration (such as EAI) as well external integration (commonly referenced as B2Bi). In addition, we'll look at the "political landscape" because organizational considerations can also affect the choice of integration technologies.

The typical enterprise integration environment has the following characteristics:

1. ***Multiple applications***: Including both packaged applications and those developed in-house.
2. ***Different ways to access the functions and data controlled by these applications (e.g., database, API, flat file, etc.) and varying levels of support for integration:*** Some applications may have been developed with the assumption that they will never have to be integrated with the rest of the enterprise.
3. ***Multiple platforms and operating systems***: Everything from mainframes to desktops.
4. ***Multiple component models:*** An enterprise will have applications built using CORBA, J2EE, or DCOM that don't "automatically" interoperate.
5. ***Multiple models for handling security***

aspects: Data confidentiality, user authentication, authorization, etc.

6. ***Multiple models for handling application transactions***: Two-phase commit, open nesting of transactions, etc.
7. ***Varying levels of application availability:*** Some applications are expected to be up and running continuously, while others may have intermittent availability.
8. ***Different data formats (both within the enterprise and across trading partners):*** XML, native text, native binary, etc.
9. ***Different network protocols:*** IIOP, MQ, HTTP(S), etc.
10. ***Systems with difficult and expensive problems of software version control and upgrade management***: Some systems may require an all-or-nothing upgrade, while others may allow piecemeal upgrades of system components.
11. ***Different performance and response requirements***: Some systems must provide near real-time response, while others may take days to complete (or much longer in some cases, e.g., a system that wants to be notified whenever a partner contract expires may deploy a "logical workflow" solution that runs for years!).
12. ***The political landscape***: Note that it may be difficult to get different teams to agree on a common component model, and that different teams may have very different areas of technological expertise. Thus there are costs associated with reaching and implementing a cross-organizational technology agreement – e.g., retraining some or all of the developers.

Given this enterprise integration environment, the next order of business is to identify the precise subsets of the problems, such that a unique solution can be efficiently applied to each subset. The goal therefore is to determine the right tools for a particular job, instead of trying to apply the same tool for every job.

Integration Problem Domain and Architectural Requirements

The previous section highlights the nature of the integration problem, which implicitly places certain architectural constraints on the solutions. These constraints are listed below (numbers in parentheses refer to the integration environment characteristics described above).

It is unlikely that a single solution would fulfill the entire list of requirements and constraints. Therefore, these constraints should not be interpreted as mandating a single comprehensive solution. Products will eventually become available to address more narrowly focused problems. But while a given integration problem is unlikely to require all the capabilities listed below, a comprehensive integration toolkit will exhibit the following characteristics:

- ***Capable of integrating different systems via their existing interfaces, preferably with "zero-touch" (1)***. (*Note:* "Zero-touch" refers to integration solutions that require no changes to existing application code – not even relinking or recompiling.) Existing applications (e.g., "dusty decks" or packaged applications) may not tolerate any change.
- ***Not restricted to any particular mechanisms for interacting with the application systems (2).*** It should be possible for the "glue code" that implements integration connections to use a file interface, an application API, or any other interaction mechanism.
- ***Platform and language independent (3).***
- ***Neutral with respect to the structure of the implementation behind the endpoints (4, 5, 6, 9).*** The integration solution should be neutral with respect to the component model (and the security model, the transaction model, etc.) of the application behind an endpoint.
- ***Independent of endpoint availability models (7, 11).*** Only one end of the interaction may be available at a given time. There should be no restriction on how long the interaction should take.
- ***Adaptable to any data format (7).*** Support for a neutral data description language is necessary in order to adapt to any data format. In addition, the data description language should be:
  - Rich enough to support the semantics and constraints of the different endpoint data formats
  - Easily translatable to the different endpoint data formats. A self-describing language is the key to making the translation easy
  - Independent of the transport protocol used for communicating with the endpoints (8).

Central to most of these requirements is a common theme of neutrality. That is, the integration solution should be neutral to platforms, languages, application component models, transaction models, security models, transport protocols, invocation mechanisms, data formats, endpoint availability models, and so on.

In other words, the integration solution largely depends upon an abstraction layer that exposes, in a neutral format, the data and service interfaces of different endpoints participating in an integration scenario. Supporting the other aspects of neutrality, such as language neutrality, transport neutrality, etc., also requires support for a neutral data format and neutral service interface specification for different languages, platforms, transports, etc. Once such an abstraction layer is established – to expose in a common language the service and data interfaces of different endpoints – the next challenges are related to interaction patterns between the endpoints.

The interaction patterns are mainly related to determining the semantics of the data passing through the endpoints, the granularities of the service interfaces offered by the endpoints, the contextual information that needs to be exchanged along with data, and so on.

These requirements impose constraints on the interaction pattern. Note that the details of the applications behind the endpoints need to be hidden from the integration solution. In order to hide the internal application model, the endpoints must expose coarse-grained interfaces that reduce the number of interactions. The reduced number of interactions requires passing semantically richer data structures through the interfaces, and deriving finer-grained data from these rich data structures. These rich data structures are commonly called business documents, and the interaction between disparate systems via exchange of the business documents is commonly called document-style interactions.

The other style of interaction (called RPC style) typically involves passing a small number of individual data items in multiple requests, and synchronously getting a small number of reply data items in return. In this environment, the decision to invoke a particular synchronous call, and the data to be passed to the call, depends heavily upon the context, which is defined by the previously invoked RPC calls and the data returned by them.

A document-style communications approach organizes data within a collection, called business document, and makes far fewer invocations compared with the RPC style. The business document contains all of the information required for business processing and typically contains far more data than the amount passed in RPC-style parameters. While a document processing request/response could be synchronous, an asynchronous approach is far more common.

In analyzing the requirements of neutrality with regard to the application model behind the endpoint, it's clear that the desirable interaction pattern in such an enterprise integration environment is that of the coarse-grained document style.

The document-style interaction pattern is also a natural choice for supporting different availability models. In document-style interaction, all the necessary information for invoking a service is encoded in stand-alone business documents, allowing applications to have life cycles independent of each other, or at least letting each one function normally for prolonged periods of time, without communicating with each other. This makes it possible for clients to make requests, and for a service to consume and process requests, independent of whether the client or service is immediately available.

Theoretically, the RPC style could also utilize a communication model where the client and server operate independent of each other's availability. However, this is often not useful in practice because applications using the RPC style typically depend upon an immediate response so that the application can decide its next step.

Another term that is commonly used in describing and differentiating the nature of interactions between endpoints is loosely coupled vs. tightly coupled. The coupling between any two systems essentially indicates the degree of dependency they have on each other. The degree of coupling implies some level of cooperation, as the two communicating systems must have a certain level of agreement and the degree of coupling depends upon the nature of that agreement.

“...the most important consideration for choosing a particular technology is its suitability for a particular problem/solution domain”

If the clients are required to program their logic against a specific API and a specific application model exposed and controlled by the service, then the dependency (and hence the coupling) between the client and service is tighter. Such tightly coupled systems typically involve invocation of multiple fine-grained APIs, which essentially visit different parts of the application.

As a result, tightly coupled interactions largely depend upon a general acceptance of the component model on which the application is designed. In these systems, the connections that a client makes with the service are dedicated to specific purposes and are therefore not reusable in most cases. Integration using such models tends to result in brittle and inflexible systems in the sense that a change in the use of interface on one side requires modification on the other side.

On the other hand, loosely-coupled systems do not bind to each other using application-specific interfaces. Instead, these systems make use of abstract message definitions to mediate their binding with respect to each other. For example, the interfaces of loosely coupled systems focus on the message definitions instead of method signatures. This supports general-purpose message definitions such that the application code can independently handle the complexity of processing specific message instances, which makes the interfaces reusable.

While a tightly coupled approach offers tremendous runtime efficiency (because the common component model can be directly exploited for low-level use case scenarios), a loosely coupled approach provides flexibility to support different high-level use cases.

Conclusion

In summary, the requirements described here demand an integration technology that is neutral with respect to object models, interaction styles, and transports, and which can provide very high interoperability. The interoperability is determined by the way the exchanged documents are defined, produced, and consumed. The integration technology needs to support defining the data and service interfaces in a neutral manner – and must build on this semantically rich layer of data and service descriptions so as to support document-style, loosely coupled interactions between the endpoints.

Note that in addition to the documents themselves, the endpoints pass additional metadata such as security context. Therefore, the solution must also take care of packaging and exchanging such context information in an interoperable manner.

These interoperability considerations affect the standardization of the packaging, as well as the standardization of the actual exchange of packaged messages (in other words, both the document and metadata). Again, it should be possible to use any transport for actual exchange of the messages; the choice of the transport is based primarily on the transport facilities required by a particular use case, such as reliable messaging, message brokering, etc.

• • •

The second article in this series will describe how Web services address the architectural requirements of the integration problem domain.

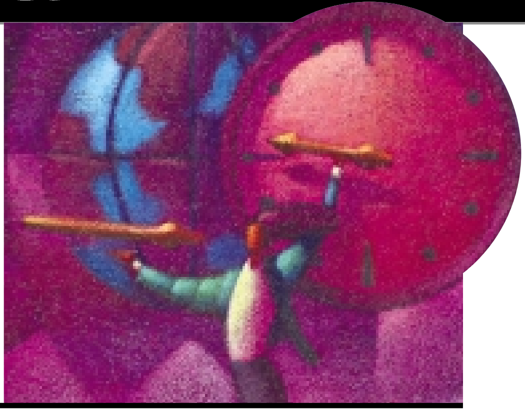
PricewaterhouseCoopers

www.pwcglobal.com.com

Written by Nikhil Patil

Document-Based Web Services Using JAXM

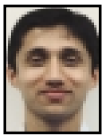
A good, but evolving, solution



RPC-style Web services aim to expose a business object as a Web service interface described by WSDL (Web Services Description Language). On the other hand, document-based Web services are based on the exchange of XML documents between two parties. The Web service receiving the document is responsible for performing actions based on the content of the XML document. The benefits of using a document-style Web service are:

- They facilitate exchange of self-describing documents that have a business context (ebXML) instead of exchange of data structures that reflect application interfaces.
- Document-based Web services support synchronous and asynchronous interactions.
- Since changing a few fields in a document doesn't break the contract between the two parties as changing the application interface would, document-based Web services provide better insulation from changes to the underlying service.

AUTHOR BIO:



Nikhil Patil has worked in the software industry for over six years, and is a product manager at Cysive, Inc., where he works on developing the strategic vision for the Cymbio Interaction Server. A Sun Certified Java Developer, Nikhil has an MS in industrial engineering and a BE in mechanical engineering.

NPATIL@CYSIVE.COM

JAXM (Java API for XML Messaging) is a Java Community Process project (JSR 67) that defines a Java API for exchanging business documents. JAXM aims to provide a SOAP-based infrastructure for building, sending, and receiving messages. It provides an abstraction for transport protocols (HTTP, SMTP, and FTP) and business messaging protocols (ebXML). The actual protocols used for conducting electronic business depend upon the implementation of JAXM.

Complex e-business applications are a combination of synchronous and asynchronous interactions. Consider, for example, Acme Consulting Company. Acme allows its consultants to bill their clients by using a Web interface (see Figure 1). The interaction between the consultant and the Web application is essentially synchronous in nature. Acme's Web application interfaces with its billing system. On receiving the billing information, the billing system performs a series of operations (e.g., getting customer information from the ERP system) that can take a considerable amount of time. To improve the user experience by reducing response times, an interaction like the one between the Web application and the billing system can be modeled as an asynchronous interaction. In this article, we'll use the Acme example (see Figure 1) to demonstrate how JAXM can build document-based interactions that can be either synchronous or asynchronous.

JAXM Basics

Figure 2 illustrates the fundamental elements of the JAXM architecture.

JAXM Service

A JAXM service consumes JAXM messages sent by a JAXM client. To develop a JAXM service, developers extend the class javax.xml.messaging.JAXMServlet and im-

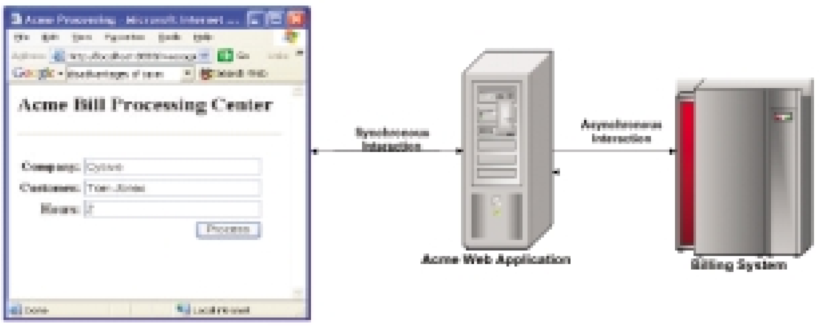


FIGURE 1 Sample application

plement either javax.xml.messaging.OnewayListener or the javax.xml.messaging.ReqRespListener interface. The choice of interface implemented depends upon whether the interaction between the client and the server is asynchronous (Oneway Listener) or synchronous (ReqRespListener) in nature.

JAXM Client

Clients exchange messages with JAXM services. JAXM clients can either directly interact with a JAXM service or go through a JAXM provider. JAXM clients that interact directly with a JAXM service can only participate in synchronous interactions. JAXM clients that use a messaging provider can participate in asynchronous as well as synchronous interactions with the JAXM service.

JAXM Messaging Provider

A JAXM messaging provider is responsible for managing the routing of messages for a JAXM client. Besides providing a degree of separation between a JAXM client and service, a provider can also offer additional services like reliable messaging, security, and transaction support. Currently, clients that want to interact asynchronously with a JAXM service have to use a provider.

JAXM Message

All JAXM messages conform to the SOAP 1.1 and SOAP with Attachments standards. JAXM messages also support the concept of higher-level protocols like ebXML through the concept of *messaging profiles*. A messaging profile defines the messaging contract between two parties. A JAXM client and service that share a message profile like ebXML can conduct business by exchanging ebXML messages.

Developing Asynchronous Interactions

JAXM clients that want to interact asynchronously with a JAXM service have to use a messaging provider, and must run in a servlet container or an EJB container as a message-driven bean. This restriction allows the messaging provider to notify the client when the JAXM service has finished processing their request.

In our example, the controller servlet (acme.ControllerServlet) is a JAXM client that asynchronously exchanges billing information with the billing system (see Figure 3) through a JAXM service (acme .BillProcessingServlet).

The controller servlet creates a connection to a JAXM provider during initialization as shown in Listing 1. The controller servlet uses the default implementation of the JAXM provider that is bundled with the JAXM reference implementation. Clients can also retrieve JAXM providers that are bound to a JNDI context.

Before sending a message, the controller servlet selects a JAXM profile that is supported by the provider. A list of supported profiles can be obtained from the ProviderMetaData class instance that is obtained by invoking the getMetaData method on the ProviderConnection class (see Listing 2).

In Acme's case, the controller servlet uses the SOAP-RP profile provided by the reference implementation. SOAP-RP provides an implementation of the Web Services Routing Protocol that supports messaging between applications through an intermediary, like a JAXM provider.

Next, the controller servlet uses a custom MessageFactory to create a message that supports the SOAP-RP profile.

```
SOAP-RPMessageImpl msg = (SOAP-
RPMessageImpl)mf.createMessage();
```

After adding the necessary elements to the message body (see Listing 3; Listings 3–8 can be found online at www.sys-con.com /webservices/sourcec.cfm), the controller servlet invokes the send method on the Provider Connection. The underlying XML message sent by the client is shown in Listing 4.

The send method assumes a one-way asynchronous communication between the client and the service. Invoking the send method by itself doesn't guarantee successful delivery of the message to the JAXM provider. The underlying JAXM provider implementation is responsible for delivering the message successfully to the JAXM service.

“The actual protocols used for conducting electronic business depend upon the implementation of JAXM”

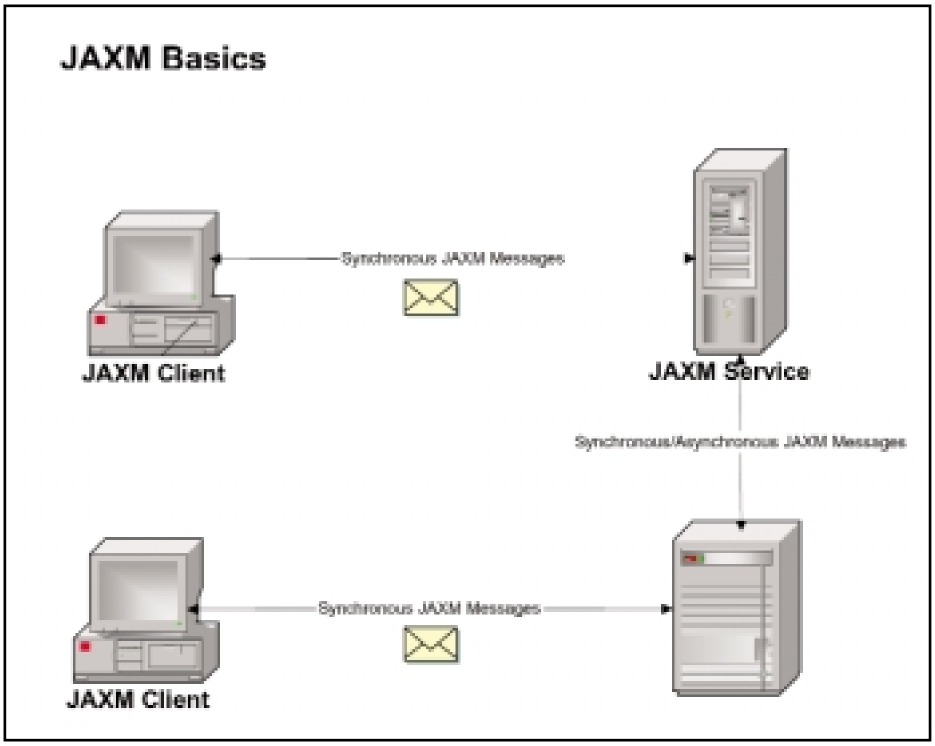

JAXM Basics
Synchronous JAXM Messages
JAXM Client
JAXM Service
Synchronous/Asynchronous JAXM Messages
Synchronous JAXM Messages
JAXM Client

FIGURE 2 Elements of JAXM Architecture

The JAXM messaging provider asynchronously routes the SOAP message to the JAXM service implemented by the class acme.BillProcessingServlet. On receiving the message the JAXM service extracts information about the bill from the message (see Listing 5). This information is then passed on to the billing system. Note that since this is an asynchronous interaction, the JAXM service doesn't return any response.

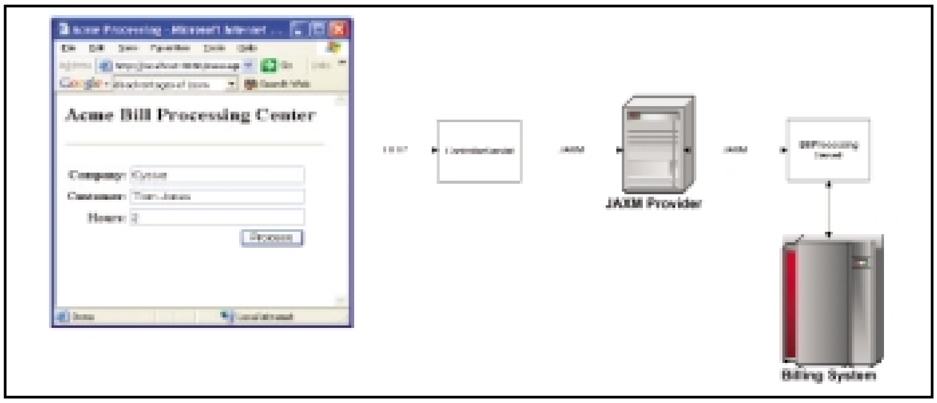


FIGURE 3 **Asynchronous JAXM service**

Developing Synchronous Interactions

Although synchronous interactions are typically thought of as synonymous with RPC-style Web services, there are situations where document-based Web services make a better case for synchronous exchange of documents. Take the example of two parties that want to synchronously exchange ebXML documents. An RPC-style Web service that exposes a system's object model through WSDL may not be capable of exchanging ebXML documents, whereas a document-based JAXM service which is not bound to by an object model can easily lend itself to this situation.

In our example, consultants can use Acme's Web application to search for a list of bills by the name of the company (see Figure 4). The controller servlet (JAXM client) conducts the search on behalf of the consultants through a synchronous message exchange with a JAXM service (acme.BillingInformationServlet).

The controller servlet uses the default MessageFactory class to create an empty SOAP message and then populates the SOAP message body with required elements (see Listing 6).

After constructing the message, the controller servlet uses class javax.xml.soap.SOAP ConnectionFactory to create a connection and send the message as shown below:

```
SOAPConnection connection =
SOAPConnectionFactory.newInstance().crea
```

SHOP ONLINE AT **JDJSTORE.COM** FOR BEST PRICES OR CALL YOUR ORDER IN AT **1-888-303-5282**

BUY THOUSANDS OF PRODUCTS AT GUARANTEED LOWEST PRICES!

GUARANTEED BEST PRICES
FOR ALL YOUR
WEB SERVICES
SOFTWARE NEEDS

JDJStore.com

N-ARY
$299.00 n-ary Ticket System v2.0

This installable application is a clean-cut Web-based tool that enables you to log & track important information. It can be used as an intranet or extranet product, giving your customers access to see how their issue is progressing. A unique number ID is assigned to every ticket. The system is extremely flexible and simple to use and can be utilized in numerous different situations. With more features than before, The Ticket System can be used for any number of great applications, which you can tailor as much or as little as you like.

GREENPOINT INC.
$1,350.00 WEB CHARTS 3D VER. 4.7

WebCharts 3D is a state-of-the-art visualization package designed for the professional Web developer. It provides general purpose and specialized 2- and 3-dimensional charts, grids, and heat maps that can be delivered as server-generated interactive images (PNG, GIF, JPG, SWF, SVG, WBMP) or applets to browsers and mobile devices.

CHUTNEY TECHNOLOGIES
$755.25 Chutney SOAP+ Toolkit

The Chutney SOAP+ Toolkit is the industry's first Web services monitoring and optimization toolkit. The Toolkit fills the functionality gaps in the leading SOAP development libraries by providing the ability to accurately pinpoint Web services bottlenecks and eliminate them through optimization techniques such as caching. The Chutney SOAP+ Toolkit acts purely as a supplement to these libraries, so no changes to the existing application logic are required. Flexible in its feature set, the Toolkit provides value to applications serving as either Web service consumers or providers.

ALTOWEB
$2,500.00 Application Platform Release 2.8

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

JALERTS
$1,495.00 JAlerts Deployment (ICQ)

If you need to send real-time Instant Messaging Alerts and Notifications to your employees or customers over the public IM services such as AIM, MSN, ICQ and Yahoo than JAlerts is your solution. Based on Zion Software's popular JBuddy SDK technology, JAlerts provides an even simpler solution for sending real-time messages.

INFRAGISTICS
$795.00 JSuite 6.0 Basic

Lightweight components with heavyweight features, the JFCSuite v2 extends the standard Java Foundation Classes (JFC) to provide a series of feature-rich UI JavaBeans components.

W W W . J D J S T O R E . C O M

OFFERS SUBJECT TO CHANGE WITHOUT NOTICE

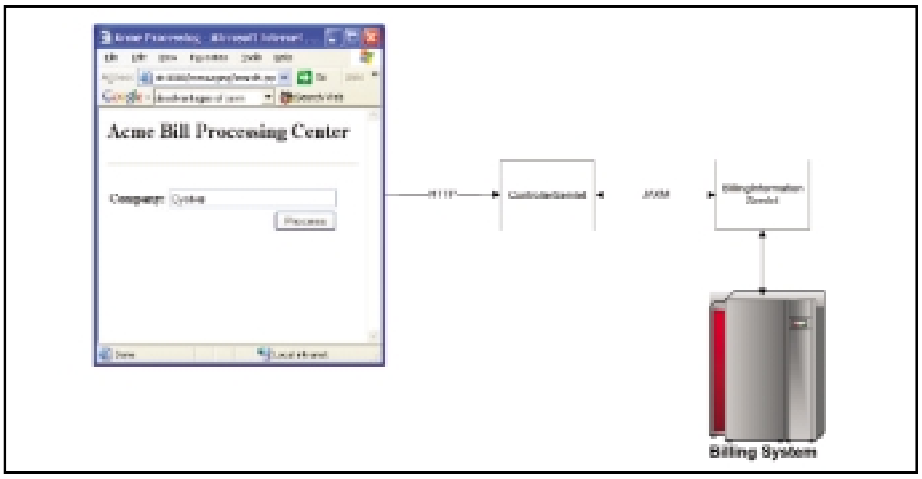


FIGURE 4 Synchronous message exchange

```
teConnection();
java.net.URL endpoint =
 new java.net.URL("http://local-
host:8080/messaging/findBills");
// Send the message
SOAPMessage responseMessage = connec-
tion.call(message, endpoint);
```

Note the difference between the call method of the class SOAPConnection and the send method (see Listing 3) of the Provider Connection class. The call method accepts an instance of java.net. URL, which directly points to the JAXM service. The send method of the ProviderConnection class assumes that the address of the JAXM service is embedded in the SOAP message. The call method also blocks the JAXM client until the JAXM service returns a response.

The message is directly routed to the JAXM service implemented by the servlet jaxm.BillingInformationServlet.This servlet extends the class JAXMServlet and implements the ReqRespListener interface. As shown in Listing 7, the BillingInformation Servlet extracts the search criteria from the incoming SOAP message, interfaces with the Acme billing system to conduct a search and returns a SOAP message containing the search results to the JAXM client.

Example Installation Instructions

To try the example used in this article, perform the following steps:

1. ***Download and install Sun's Java Developers Web Service Pack (JDWSP).*** The software can be found at http://java.sun.com/Webservices/downloads/webservicespack.html.
2. ***Unzip file Messaging.jar.*** This will create a folder called messaging on your hard drive.
3. ***Open the file build.properties in the messaging folder.*** Change the property jdwsp.home to point to the directory in which the JDWSP was installed.
4. ***Assuming that you have Ant installed on your machine,*** run the command Ant (from command prompt or a shell prompt) from the folder messaging.
5. Start the Tomcat servlet engine bundled with JDWSP.
6. Using the URL http://localhost:8081/jaxm-provideradmin/index.jsp, log into the JDWSP administration tool.
7. Add the following endpoint mapping to the SOAP-RP profile .

```
Uri: http://xyz.acme.com/processbill
URL: http://127.0.0.1:8081/jaxm-
provider/receiver/soaprp
```

8. You can submit billing information by using the URL http://localhost:8080/messaging/submitBill.jsp.
9. You can search for bills by company name by using the URL http://localhost:8080/messaging/search.jsp.

Conclusion

JAXM provides a good starting point for developing document-based Web services that can promote exchange of information between enterprises in a loosely coupled fashion through context-sensitive documents. It provides a flexible protocol, supporting both synchronous and asynchronous interactions between participants.

JAXM is an evolving specification and still doesn't address issues like:

1. ***Support for asynchronous messaging without using a message profile:*** Currently the only way to communicate asynchronously with a JAXM service is to use a message profile like ebXML or SOAP-RP profile.
2. ***Better interoperability with DOM:*** Currently, JAXM allows the developer to set the entire content of a SOAP message by invoking the setContent method of the javax.xml.soap.SOAPPart class. There is no support for inserting XML data fragments into a SOAP message.

References

- *Java API for XML Messaging:* http://java.sun.com/xml/jaxm/index.html
- *Web Services Routing Protocol:* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-routing.asp

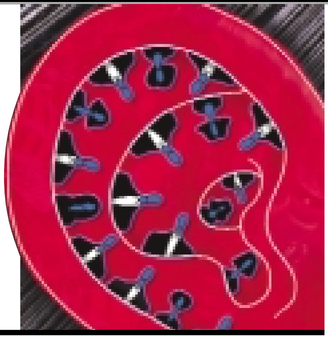
Listing 1

```
 try
 {
  pcf =
ProviderConnectionFactory.newInstanc
e();
  pc = pcf.createConnection();
 }
 catch (SOAPException e)
 {
  throw new ServletException(e);
 }
```

Listing 2

```
 String[] supportedProfiles =
metaData.getSupportedProfiles();
 String profile = null;
 for(int i=0; i <
supportedProfiles.length; i++)
 {

if(supportedProfiles[i].equals("soap
rp"))
  {
   profile = supportedProfiles[i];
   break;
  }
 }
```

Download the code at
sys-con.com/webservices



Intercepting SOAP Messages

Easy development based on Apache Axis

This article takes a look at SOAP Message handlers and how JAX-RPC supports SOAP Message handlers and its usage scenarios.

First, some of the terminology: a SOAP Interceptor takes raw SOAP message as input and, before processing the message, modifies the SOAP message or adds some additional information to it, and then returns the SOAP message as output.

AUTHOR BIOS:

Aravilli Srinivasa Rao, a software analyst at Hewlett-Packard, is technical lead for the development of HP's public UDDI Registry. He is currently involved in a feasibility study of the projects in the mobile space to implement web services. Aravilli holds a master's degree in computer application.
SRINIVASA.RAO.ARAVILLI@HP.COM

Madan Mohan, a senior software engineer at Hewlett-Packard, has several years of experience in J2EE, Web services, and testing. He is currently involved in the development and testing of J2EE and Web services applications. Madan holds a bachelor's degree in engineering.
MADANM@HP.COM

A *SOAP Message handler* is a mechanism that intercepts the SOAP messages at the client level as well as at the server level. SOAP Intermediary intercepts the SOAP message between the client and server.

SOAP Header processors deal mainly with the header part of the SOAP payload. A SOAP Header typically contains authentication information, a transaction ID, a digital signature, or some routing information.

SOAP Message Handlers

A SOAP Message handler intercepts the SOAP messages between the client and the server and gets access to the SOAP message that represents either SOAP request or SOAP response. Message handlers are tied to the Web service endpoints and are used to provide additional processing mechanisms when invoking a service using RPC. A SOAP Message handler processes the SOAP Header blocks as part of the processing of an RPC request or response.

SOAP Message Handler Usage Scenarios

SOAP Message handlers are used for many scenarios. Some typical handlers are:

- To do encryption and decryption on the client and service side for exchanging messages securely
- To provide Access-Control to Web services
- To provide auditing, logging, metric collection, caching, and transformation of Web services
- To provide content-based routing
- To provide transaction management

JAX-RPC Message Handler

JAX-RPC defines the following interfaces/class in order to support the SOAP Message handler.

Handler

A handler interface provides methods for handleRequest, handleResponse, and handleFault. handleRequest and handleResponse perform the actual processing work for a handler. The method handleRequest processes the request SOAP message, while the method handleResponse processes the response SOAP message. The method handleFault performs SOAP fault processing.

SOAP Message Handler

A SOAP Message handler class is required to implement the javax.xml.rpc.handler.Handler interface.

GenericHandler

The javax.xml.rpc.handler.GenericHan dler class is an abstract class that implements the handler interface. Handler developers should typically subclass the GenericHandler class unless the handler implementation class needs another class as its superclass.

In the case of Apache Axis, handler developers should subclass the BasicHandler, which is a subclass of Handler.

HandlerChain

This class represents an ordered list of handlers. An implementation class for the HandlerChain interface abstracts the policy and mechanism for the invocation of the registered handlers. The default invocation policy is to invoke handlers in the order of registration with the HandlerChain instance. Additional policies may be applied to process the SOAP Headers.

HandlerInfo

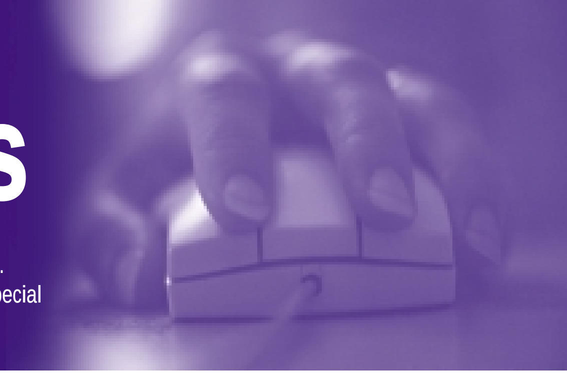
The HandlerInfo class represents the configuration data for a handler.

MessageContext

The interface MessageContext abstracts the message context that is processed by a handler in the handleRequest, handleRe-

SaturdaySessions

We understand the pressures of work and how difficult it can be to get time off. That is why we have designed this workshop to be held in one day and, as a special bonus, on the weekend, so no days off from work. **Your boss will be happy!**



Alan Williamson
JDJ Editor-in-Chief

JDJ Workshop with Alan Williamson

M	T	W	T	F	S	S

Coming to you...

April:	**May:**	**June:**
NEW YORK	BOSTON	ATLANTA
WASHINGTON, DC	TORONTO	RALEIGH

Performance > Efficiency > Reliability

This one-day intensive workshop is designed for developers who wish to increase the efficiency and reliability of their code development.

1) The day will begin by looking at the various hints and tips you can utilize at the code level to improve the quality and reduce the number of bugs you have to contend with.

2) The next part will look at Apache's Ant and how you can use this freely available tool for your own development, irrespective of your IDE.

3) Last, and most important, as the old saying goes: "You can never do enough testing." This session will look at JUnit and show you how to start building test harnesses for your code so you can begin your testing strategy.

>Performance

Java is a powerful language. While it offers a rich array of tools, the fundamentals mustn't be overlooked. Improving your code at the core layer will result in great improvements in efficiency and produce (hopefully) less bugs. We'll look at the do's and don'ts of programming and learn many hints and tips that will accelerate your Java coding.

>Efficiency with Ant

Apache's Ant is a powerful scripting tool that enables developers to define and execute routine software development tasks using the simplicity and extensibility of XML. Ant provides a comprehensive mechanism for managing software development projects, including compilation, deployment, testing, and execution. In addition, it is compatible with any IDE or operating system.

> Reliability with JUnit

A critical measure of the success of software is whether or not it executes properly. Equally important, however, is whether that software does what it was intended to do. JUnit is an open-source testing framework that provides a simple way for developers to define how their software should work. JUnit then provides test runners that process your intentions and verify that your code performs as intended. The result is software that not only works, but works in the correct way.

What you will receive...

- ✓ INTENSIVE ONE-DAY SESSION
- ✓ DETAILED COURSE NOTES AND EXCLUSIVE ONLINE RESOURCES
- ✓ JDJ CD ARCHIVE

ONLY $295*

*JDJ subscribers ($395 non-subscribers)

*GROUP DISCOUNTS AVAILABLE

To Register

www.sys-con.com/education

Call 201 802-3058

SPONSORED BY JAVA DEVELOPER'S JOURNAL

PRODUCED BY SYS-CON EVENTS

PRESENTED BY SYS-CON i-TECHNOLOGY EDUCATION sponse, or handleFault. It has a method to set the properties, which are shared across the handlers in the handler chain.

SOAPMessageContext

The interface SOAPMessageContext provides access to the SOAP message for either RPC request or response. It has methods to set and get the SOAP message.

Handler Configuration

A JAX-RPC handler may be configured and used on the service **client** as follows:

- ***A client-request handler:*** A request handler is invoked before an RPC request is communicated to the target service endpoint.
- ***A client-response handler:*** A response handler is invoked before an RPC response is returned to the service client from the target service endpoint.

A JAX-RPC handler may configured and used on a service **endpoint** as follows:

- ***A server-request handler:*** On the service endpoint side, a request handler is invoked before an RPC request is dispatched to the target service endpoint.
- ***A server-response handler:*** A response handler is invoked before an RPC response is communicated back to the service client from the target service endpoint.

Let's take a look at how to develop a SOAP Message handler and how to configure the handler at the service endpoint as a request handler using the Apache Axis JAX-RPC implementation.

Using SOAP Message Handlers in Web Services

First, let's assume you're working on an application where you need to send secure SOAP messages to the Web service. In our example, we're developing a purchase order processing system where customers submit purchase order requests and the system processes the purchase orders. Each customer is assigned a private/public key pair that is installed in the client machine in a keystore, which is a database for keys and certificates. The customer signs each purchase order document using the private key and passes the public key and certificate details along with the document. Customers submit the signed purchase order document using a Web service to the supplier.

On the supplier site, each XML Signature is verified and if the identity of the submitter is accepted the purchase order is accepted for further processing. A SOAP Message handler is used in order to verify the XML Signature on the supplier side.

Let's consider how to develop the client application to digitally sign and send the purchase order document and how to develop the server-side application to verify the signature and process the document. We'll use Apache Axis.

Develop Client Application

Develop a Client application with the following:

- Purchase order generation
- Sign the purchase order document and send it to service endpoint

Generate the purchase order (see Listing 1; the Java code for this article, contained in Listings 1–5, can be found online at www.sys-con.com/webservices/sourcec. cfm) as shown below. Assume that the Shipping and Billing addresses are the same.

```
<purchaseOrder orderDate="2003-01-20">
  <shipTo country="IN">
    <name>ARAVILLI    </name>
    <street>Cunningham road    </street>
    <city>Bangalore    </city>
    <zip>560052    </zip>
   </shipTo>
   <Items>
    <item partNum="872-AA">
     <productName>Inkjet-Printer
</productName>
     <quantity>5     </quantity>
     <USPrice>148.95     </USPrice>
    </item>
   </Items>
  </purchaseOrder>
```

Sign the purchase order document (see Listing 2) using the private key and pass the certificate information. It signs only the purchase order element tag instead of signing the entire SOAP Body. The signature element looks like Listing 6.

Now create a SOAP message with the purchase order element in the body of the SOAP message and signature in the SOAP Header block. Add the service name and method as part of the SOAP Body in the message (see Listing 3 for Java code and Listing 5 for the entire SOAP message). The generated SOAP message looks like Listing 7.

To generate this SOAP message, we used a message-style Web service instead of RPC.

Develop the Server Application

Develop the server application:

```
public class Service
{
    public String processPOMethod ()
    {
        // Actual processing done here
    }
}
```

Develop the Message Handler

Develop a class that extends from BasicHandler of Apache Axis and override the invoke method. Apache Axis defined BasicHandler as an abstract class that extends from Handler, so the handler developer should subclass his pr her handler from BasicHandler class. The Invoke method receives MessageContext, which contains the incoming SOAPMessage. From the SOAP message extract the Signature element using XPath APIs and verify it against the SOAP Body. If the verification succeeds, the handler passes the message (removes the signature) on to the Web service (purchase OrderService) server itself. If the verification fails then the handler raises an exception, which causes a SOAP fault to be returned (see Listing 4 for Java code).

Deploy the Service and Configure the Handler

Configure the Handler

Configure the handler using the Handler element in the Apache Axis Web service deployment descriptor. This element has two attributes: the name for the handler and the type of handler. Handler type refers to the Java class that implements the handler. The Handler element contains several child <parameter> elements, which can be used to provide the configuration details for the handler:

```
<handler name=" signature"
type="DecryptionHandler">
<parameter name="filename"
value="MyService.log"/>
</handler>
```

Handler chain is a collection of one or more handlers:

```
   <chain  name = "my_own_chain" >
```

```
 <handler name=" Handler1"
type="Type1">
<parameter name="param1" value="Value1"/>
</handler>
 handler name="Handler2 " type="Typ2">
<parameter name="param2" value="value2"/>
</handler>
</chain>
```

In order to reuse the handlers, define them separately and reference them in the chains.

```
<handler name=" Handler1"   type="Type1">
<parameter name="param1" value="Value1"/>
</handler>
<handler name="Handler2 "   type="Typ2">
<parameter name="param2" value="value2"/>
</handler>
<chain  name = "my_own_chain" >
<handler type =" Handler1"  >
<handler type =" Handler2"  >
</chain>
```

Axis supports the Flow element to bind a particular handler or chain of handlers to a request or response. A Flow defines the sequential invocation of handlers and handler chains. The supported flows in Axis are:

- ***Request Flow:*** Describes inbound messages
- R***esponse Flow:*** Describes the outward flow of a message
- ***Fault flow:*** Describes the fault processing flow to be executed for SOAP Fault

Our example, related to Request Handler, uses the Request Flow element to bind the handler.

```
<requestFlow>
<handler type="signature"/>
</requestFlow>
```

We have defined handler element separately and referenced it in the requestFlow. It may also be defined in the request Flow element itself:

```
 <requestFlow>
 <handler name=" signature"
type="DecryptionHandler">
<parameter name="filename"
value="MyService.log"/>
</handler>
</requestFlow>
```

Configure and Deploy the Service

Configure and deploy the service using the Axis WSDD file as shown in Listing 8. Once deployed, the service invokes/tests the service and handler using the client application.

Conclusion

As you can see, it's not difficult to develop the SOAP Message handlers using Apache Axis; the handlers can be used in various ways. Axis used handlers extensively in the design of the Axis Engine. Refer to the Axis Architecture document for more information about the handlers usage.

References

- *Apache Axis*: http://xml.apache.org/axis
- *JAX-RPC*: http://java.sun.com/xml/jax rpc
- *SOAP*: www.w3.org/TR/SOAP

Listing 6

```
<SOAP-SEC:Signature>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
     <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
     <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
     <ds:Reference URI="#purchaseOrder">
      <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>2jmj7l5rSw0yVb/vlWAYkK/YBwk=
</ds:DigestValue>
   </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>

ZW7stqLKyeLlHxJ611C7LLLcw96ArOVnNNw/J/6e+TeDDeaa1K1EtA==</ds:Si
gnatureValue>
    <ds:KeyInfo>
    </ds:KeyInfo>
   </ds:Signature>
  </SOAP-SEC:Signature>
```

Listing 7

```
<soapenv:Envelope soapenv:actor="some-uri"
soapenv:mustUnderstand="1"
// namespace declaration >
<soapenv:Header>
   <SOAP-SEC:Signature </SOAP-SEC:Signature>
  </soapenv:Header>
<soapenv:Body>
   <ns1:processPOMethod
xmlns:ns1="http://localhost:8080/POService"/>
  <purchaseOrder orderDate="2003-01-20">
 </purchaseOrder>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 8

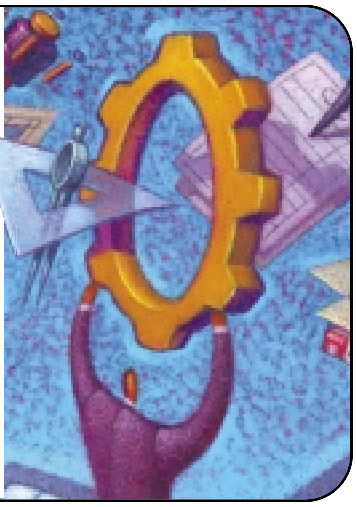
```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"

xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <!-- define signature handler configuration -->
 <handler name=" signature" type="DecryptionHandler">
<!--define the file to log the exceptions/faults -- >
<parameter name="filename" value="MyService.log"/>
 </handler>

 <!-- define the service, using the log handler we just
defined -->
 <service name="http://localhost:8080/POService"
provider="java:RPC">
  <requestFlow>
   <handler type="  signature"/>
  </requestFlow>
<!—specify the class name and its methods to the purchase
order service
  <parameter name="className" value="POService"/>
  <parameter name="allowedMethods" value=" processPOMethod
"/>
 </service>
</deployment>
```

Download the code at
sys-con.com/webservices

Written by Thomas Schaeck & Stefan Hepper

Portal Standards for Web Services

Moving toward interoperability



Portlets are visual components that make up a Web page residing in a Web portal. Typically, when an end user requests a personalized Web page, multiple portlets are invoked when that page is created. An example is a news/financial portal that displays a single page that includes updated financial news, a report on how the stock market is doing, and the latest information on stocks of interest to the end user. Each component consists of one or more portlets.

Portlets rely on APIs to communicate with the portal and access various types of information, such as a user profile. The lack of standards has led portal server platform vendors to define proprietary APIs for local portal components and for invocation of remote components. This creates interoperability problems for portal customers, application vendors, content providers, and portal software vendors.

The Java Portlet Specification JSR 168 and Web Services for Remote Portals(WSRP) standards are being developed to overcome these problems, providing interoperability between portlets and portals, and between portals and user-facing Web services.

The Java Portlet Specification establishes interoperability between portlets and portals. All portlets written to the JSR 168 Portlet API will run on all compliant portal servers. This API will cleanly separate portlets from the surrounding portal server infrastructure so that the portlets can run on different portal servers, just as servlets can run on different application servers.

Similarly, WSRP will enable interoperability between portals and WSRP-compliant Web services for portals. WSRP services are presentation-oriented, user-facing Web services that plug-and-play with portals or other applications. They are designed to let businesses provide content or applications in a form that does not require any manual adaptation. Portals can easily aggregate WSRP services without programming effort.

Because WSRP includes presentation features, WSRP service providers can determine how end users see their content and applications, and to what degree adaptation, transcoding, and translation might be allowed. WSRP services can be published into public or corporate service directories (Universal Description, Discovery, and Integration; UDDI), where portals that want to display their content can find them easily.

Using WSRP, portals can easily integrate content and applications from internal and external content providers. A portal administrator simply picks desired services from a list and integrates them.

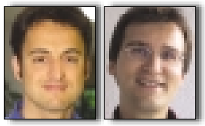
The WSRP standard will define a Web services interface using Web Services Description Language. The standard lets WSRP services be implemented in different ways, whether as a Java 2 Platform, Enterprise Edition (J2EE)–based Web service, a Web service implemented on the .NET platform, or a portlet published as a WSRP service by a portal. The standard enables use of generic adapter code to plug any WSRP service into intermediary applications rather than requiring specific proxy code. This will allow implementation of WSRP services on any Web services–capable platform, be it J2EE or .NET. The WSRP technical committee plans to have Version 1.0 ready by the middle of this year.

The Java Portlet API and WSRP will be able to cooperate in a beneficial way. WSRP services could be integrated in portals through portlet proxies written to the Java Portlet API. Conversely, portlets could be wrapped in and published as WSRP services.

Once a portlet entry is listed in the UDDI directory, other portals can find and bind to the referenced WSRP service. To make a WSRP service available as a portlet, the portal's administration may create an entry in the local portlet registry with the information obtained from UDDI. For example, once the entry is in the local portlet registry, users might select it and put copies of it on their pages. When a portlet proxy is invoked during page aggregation, the portlet proxy will generate a Simple Object Access Protocol (SOAP) request and send it to the WSRP service. Then it receives the SOAP response from the WSRP service and provides the result to the portal.

Conclusion

The Java Portlet API and WSRP standards will enable interoperability of portal servers; local portlets; and remote, interactive, user-facing Web services, respectively. The ultimate goal is that a large number of portlets that can run on any compliant portal server locally, as well as remote WSRP services that plug and play with all compliant portal servers, will be available, and that a portal component market comes into existence in which a large variety of application providers, ISVs, and the open-source community offer readily available portlets or visual Web services.



Author Bios

Thomas Schaeck is an architect in WebSphere Portal Server development. He has published various papers and filed 20 patents, and coauthored *Smart Card Application Development in Java* (Springer 2000) and *Pervasive Computing* (Addison-Wesley). SCHAECK@DE.IBM.COM

Stefan Hepper joined IBM in 1998 and is now engaged in the IBM WebSphere Portal Server project and is the colead of JSR 168 to standardize the Portlet API. He has lectured at international conferences, published papers, and filed patents, and he coauthored *Pervasive Computing* (Addison-Wesley). STHEPPER@DE.IBM.COM

Written by Kuassi Mensah

Web Services Enable Your Database

Turn your database into a Web services provider and consumer

"Si tu ne vas pas à Lagardère, Lagardère ira à toi!"
"Le Bossu", Alexandre Dumas

On guard! What do Web services have to do with a swordsman? Well, paraphrasing Alexandre Dumas's character, "if you don't go to Web services, Web services will come to you." Web services are pervading every layer of enterprise computing, from packaged e-business applications (e.g., ERP, CRM) to middle-tier (e.g., J2EE, .NET) and database infrastructure.

Web services promise easy access to remote content and application functionality using industry-standard mechanisms, without any dependency on the provider's platform, the location, the service implementation, or the data format.

The proliferation of structured and unstructured data and data logic in databases, the increasing momentum of XML as a data exchange format, the de facto acceptance of HTTP as a ubiquitous transport mechanism –in the context of heterogeneous environments – has aroused interest in database Web services. For example, many applications have significant amounts of business logic (such as PL/SQL packages) that run in the database. Database Web services allow organizations to leverage this existing investment.

Database Web services work in two directions: database as service provider, i.e. calling from the outside in, which lets client applications access the database via Web services mechanisms; and database as service consumer, i.e. calling from the inside out, which lets a SQL query or an application module in a database session consume an external Web Service. This article looks at the benefits of turning your database into a Web services player. I'll explore how the database can be enabled both as a Web service provider and a consumer of external Web services. I'll also discuss future database support for Web services.

Database as Service Provider

Motivation

Why would you consider exposing database operations as Web services? What if you could expose existing stored procedures as Web services? When should you consider database Web services and when should you consider middle-tier (J2EE, .NET) Web services?

Many organizations are considering exposing their existing data and data logic that runs in the database as Web services (to both internal and external audiences). Consider Amazon Web services (www.amazon.com/webservices) or Google Web services (www.google.com/apis/index.html). These let client applications discover and interact with their catalogs or search engines using standard Web services protocols (WSDL, SOAP). Generally, when considering the circumstances in which database Web services should be deployed, the rules of application partitioning across the middle tier and data-

AUTHOR BIO:

Kuassi Mensah is group product manager in the Java products group of Oracle server technologies. He has published several articles in leading developer magazines and is a frequent conference speaker.
KUASSI.MENSAH@ORACLE.COM

base should be observed. The business logic associated with Web services fits most naturally in the middle tier, and data logic in the database. This separation of concerns reduces application complexity; increases security, reuse, and data access efficiency; and localizes the impact of schema change.

Consider stored procedures as an example. Stored procedures are an essential database-programming model that allows a clean separation of data-centric logic that runs in the database from business logic that runs in the middle tier. Exposing a stored procedure as a Web service is a more cost-effective, more secure, cleaner, and more efficient choice than exposing a middle-tier component.

Architecture: The Service Provider Framework and the Service Invocation Mechanism

As part of their end-to-end Web services offerings, all major database vendors, including Oracle, IBM, and Microsoft, now provide access to their databases through Web services mechanisms (SOAP, WSDL, UDDI). Currently, they are all leveraging their existing middle-tier Web services framework. An example is shown in Figure 1.

IBM's DB2 also leverages the middle tier to handle SOAP encoding/decoding but uses a file-based approach for designating the database operations that are to be exposed as Web services. Microsoft's SQL Server utilizes the SQL Server 2000 Web Service Toolkit. A multistep process consisting of defining the database operations to be exposed as Web services, configuring a SOAP virtual name, and defining an IIS folder associating a SQLXML virtual name to the IIS subfolder is required.

Database APIs for Implementing Database Web Services

Due to the rapid evolution and maturing of Web services standards and technology, an exhaustive list of APIs and features provided by each database vendor is beyond the scope of this paper. However, I can give you a general idea of what database Web services you can expect to see:

- ***SQL queries and DML statements:*** The sky's the limit here. SQL offers amazing possibilities for retrieving, updating, and storing any data that can be held in your database, including relational, text, spatial, multimedia, and XML data. But let's not miss the point: Web services are not the new or fashionable way of submitting SQL statements to your database, nor a replacement for gateways. You should consider implementing coarse-grain services using SQL statements only when the benefits outweigh the SOAP overhead. SQL database Web services will allow client applications and lightweight SOAP enabled appliances to, for example, query a catalog or retrieve the map of a location based on the ZIP code.

- ***XML APIs:*** The integration of XML with relational databases enables storing, indexing, and retrieving semistructured and nonstructured data. Depending on the level of integration, XML documents can be stored either natively or through XML views over relational structure. XML database Web services will let client applications (i.e., desktop tools) retrieve XML documents, make changes locally, and synchronize back to a back-end database. This approach also hides the complexity of XML APIs, as well as concerns over storage and indexing, from the service consumer.

- ***Stored procedures and user-defined functions:*** Stored procedures are an essential database-programming model that allows a clean separation of data-centric logic that runs in the database from business logic that runs in the middle tier. Stored procedures can be programmed in a database-specific language, e.g. PL/SQL, or in a portable and database-independent language such as Java. The ANSI SQLJ Part 1 specifies the ability to invoke static Java methods from SQL as procedures or functions. The beauty of Web services is that you can leverage your investment in stored procedures and expose these as Web services without having to worry about the language in which the stored procedures are implemented.

- ***Database messaging and queuing APIs:*** A Web services API can be used to expose AQ operations: message enqueuing/dequeuing and notification registration; capturing and propagating database changes operations; and providing access to queue administration operations.

Facing Web Services Issues

How do database Web services address common Web services issues such as data type mapping, security, transactions, and interoperability?

- ***Data type mapping:*** Producing XML documents from database types requires some metadata. Simple SQL types are taken care of through generic or predefined

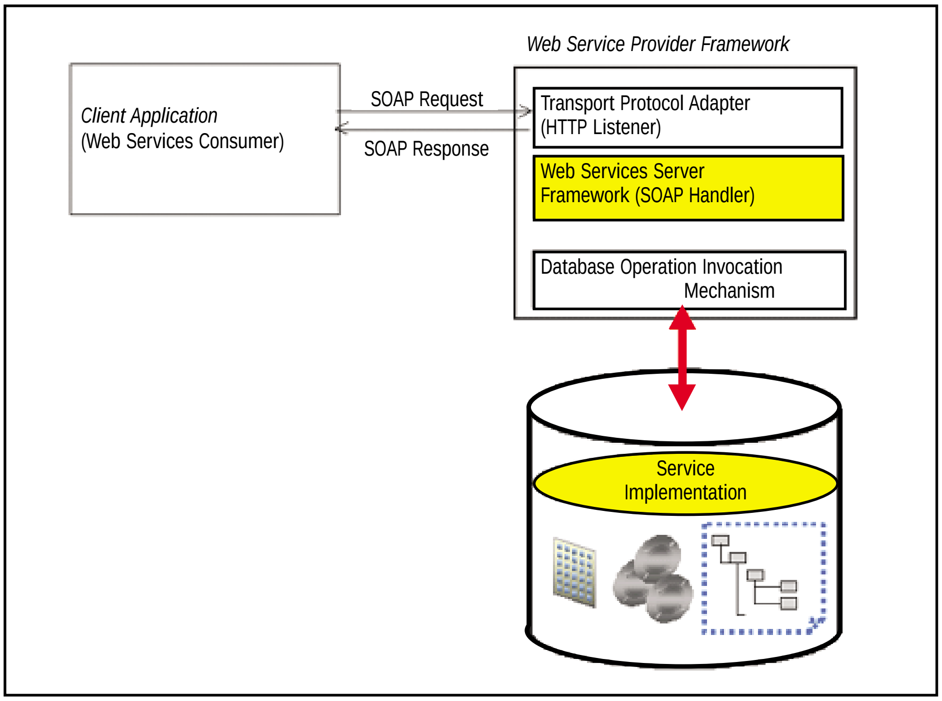


FIGURE 1 Generic database Web services framework

XML Schema. Complex database types such as Resultset, JDBC WebRowset, ADO Datasets, and other proprietary hierarchical data shapes will be mapped through specific XML Schema. The introduction of Pull parser and XML writer in JDBC and .NET will improve and simplify XML documents creation and parsing.

- ***Security:*** Since business data is the most valuable corporate asset, security management is a critical requirement. For database Web services, this requirement is addressed by leveraging the Web services security framework as well as database security mechanisms such as controlling the effective database user or schema at runtime and restricting the allowed operations through the service implementation.

- ***Transactions:*** Since standards for conducting Web services transactions across database, middle-tier, and packaged applications are still evolving, DML statements (update, insert, and delete) exposed as Web services will be automatically committed without any transaction context propagation across calls and updates will use optimistic locking.

- ***Interoperability:*** Database Web services can leverage middle-tier interoperability features. For example, Web services running in the Oracle9*i* Database leverage Oracle9*i* Application Server to benefit from built-in interoperability based on Web Services Interoperability (WS-I) and SOAPBuilder efforts.

Database as Service Consumer

Motivation - Scenario

Relational databases are extending their storage, indexing, and searching capabilities to semistructured and nonstructured data, in addition to enabling federated data, e.g. from heterogeneous and external sources including Web services. The ability to call out Web services enables databases to track, aggregate, refresh, and query dynamic data, as well as data produced on demand such as stock prices, currency exchange rate, interest rate, IRS tax tables, scientific data, and weather information. For example, a database job can be scheduled to run at a predefined frequency to invoke external Web services that return inventory information from multiple suppliers and update a local inventory database. Another example is that of a Web Crawler: a database job can be scheduled to collate product and price information from a number of sources.

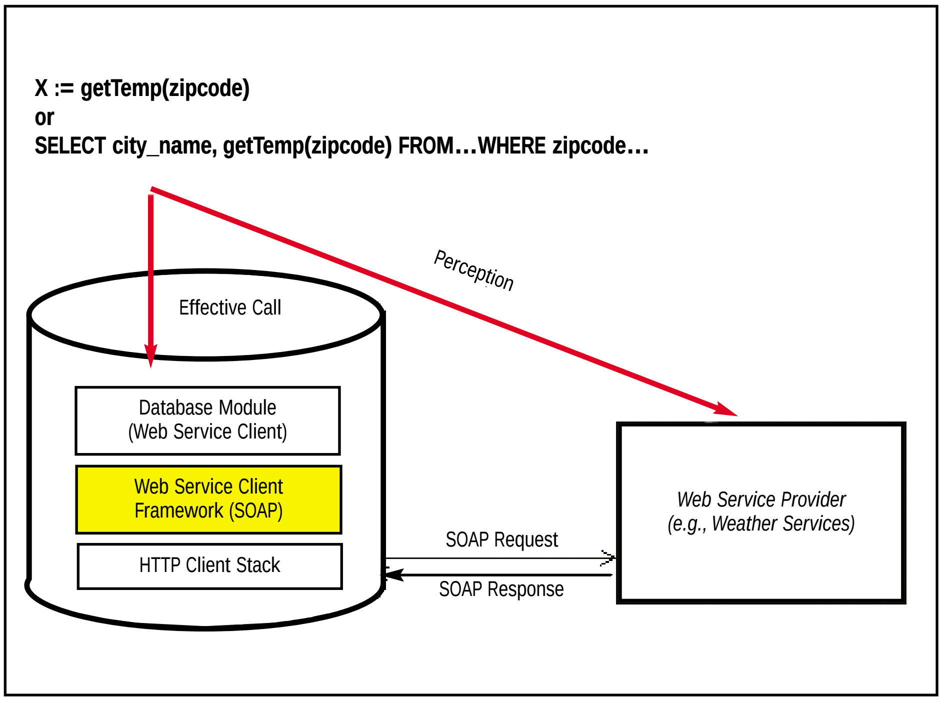

X := getTemp(zipcode)
or
SELECT city_name, getTemp(zipcode) FROM…WHERE zipcode…
Perception
Effective Call
Database Module (Web Service Client)
Web Service Client Framework (SOAP)
HTTP Client Stack
SOAP Request
SOAP Response
Web Service Provider (e.g., Weather Services)

FIGURE 2 **External Web service call-out in function call**

Architecture: The Service Consumer Framework

Consuming data from external Web services requires the following steps: building the SOAP message based on the Web service endpoint; sending the SOAP message over HTTP to the Web service endpoint; receiving the SOAP response; extracting resulting data from the SOAP message body, and dealing with faults and exceptions. Architectural issues you need to be aware of include:

- ***Non-SOAP–aware infrastructure:*** This basic approach, taken by some database vendors, requires user-defined functions to programmatically build a SOAP-like XML text (the SOAP envelope is mostly static), and send it over HTTP, parse, and decompose the SOAP response. This approach may be acceptable if you know the Web service and its format ahead of time, and if you are willing to read and interpret the service's WSDL specification yourself. It requires some programming skills, but can rapidly become time consuming and cumbersome.

- ***SOAP-aware infrastructure:*** A more flexible alternative is to use an existing SOAP client stack that understands WSDL. For example, the Oracle database allows loading Java-based libraries such as the Apache SOAP client or Sun's JAX-RPC, permitting you to dynamically interact with a Web service or utilize pregenerated client-proxy code. The SOAP stack can be refreshed simply by loading a newer version.

- ***Static Invocation:*** A pregenerated client proxy can be obtained and used by the database for invoking the Web services. A client proxy simplifies Web service invocations as it knows exactly where the service is located without looking it up in the UDDI registry and does all the work to construct the SOAP request, and marshall and unmarshall parameters. Once you have created a proxy instance, you can call the desired Web service operations on it.

- ***Dynamic Invocation:*** The static invocation is the most common invocation style. However, it requires downloading the pregenerated proxy. Dynamic invocation provides the ability to dynamically construct the SOAP request and access the service without the client proxy.

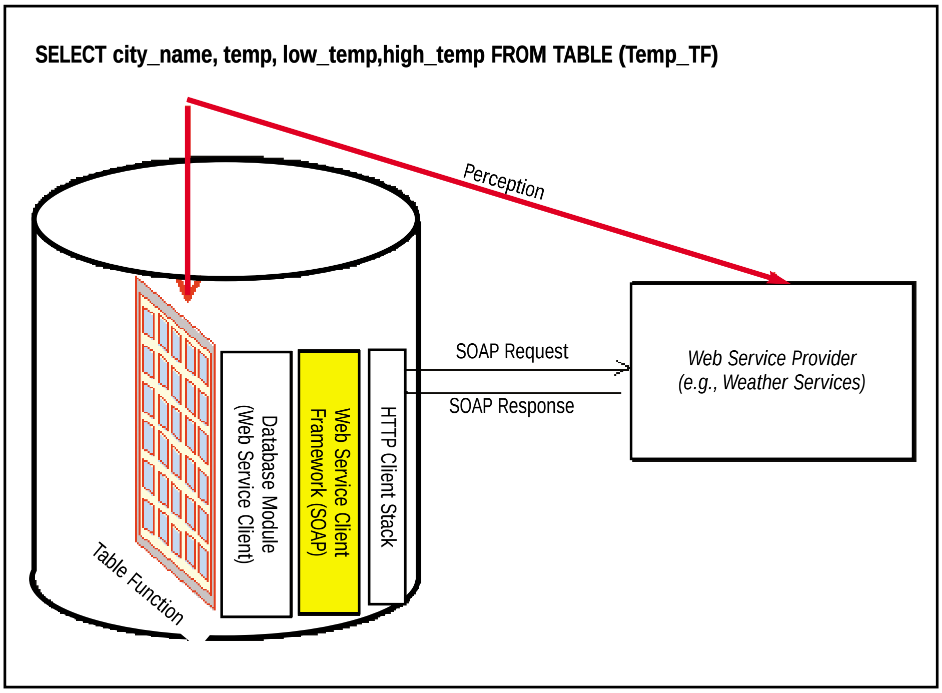

SELECT city_name, temp, low_temp,high_temp FROM TABLE (Temp_TF)
Perception
Table Function
Database Module (Web Service Client)
Web Service Client Framework (SOAP)
HTTP Client Stack
SOAP Request
SOAP Response
Web Service Provider (e.g., Weather Services)

FIGURE 3 External Web service as SQL data source

Integration with SQL Engine: SQL over SOAP

After dealing with the invocation style, you want to integrate the Web service information at the heart of the database – the SQL engine. I'll describe two mechanisms that allow you to accomplish this.

- ***Consuming External Web Service - Function Call:*** The database session invokes the Web service through a user-defined function call either directly within a SQL statement or view, or via a variable (see Figure 2).

```
X := getTemp(zipcode)
SELECT city_name, getTemp(zipcode)
FROM … WHERE …
CREATE VIEW city_temp AS SELECT
city_name, getTemp(zipcode) FROM …
```

- ***Virtual Tables – Mining Dynamic Data:*** What if multiple values are returned from the Web services? And what if you want to track and materialize the range of values returned from a series of a Web service function calls as a SQL table or view structure? How can virtual tables be used in real life? Consider the two following scenarios:
  - *Scenario 1: Calculating Aggregate Values:* An application is implemented to aggregate some temperature values for a set of cities. The application can be implemented as a store procedure that requests the current temperatures of selected cities by invoking a public Temperature Web service with the ZIP codes of the selected cities. Aggregate functions, such as max, min, and avg, can be applied to the resulting data set.
  - *Scenario 2: Collating Data:* An application is designed that collates, at predefined time intervals, the temperature for a set of cities and stores this information in a local database. This application can be implemented by a database batch job that calls the Temperature Web service with the ZIP codes of all the cities and stores the result data in a local database table.

“Database Web services leverage your existing server-side infrastructure, allowing you to use your database as both service provider and service consumer”

An architecture supporting both of these scenarios is shown in Figure 3.

What's Next for Database Web Services

Web services have received significant attention and there is a great deal of industry excitement around the opportunities afforded by them. While most of this attention has focused on middle-tier Web services, an increasing interest in database Web services has recently emerged as organizations reflect on their existing investments in data and stored procedures. So what can you expect in the future? We're likely to see increased integration with tools and frameworks, support for more complex data types, and convergence with Grid technologies. Tools will begin to support database Web services coupled with emerging Web services standards around choreography and orchestration, and we will also start to see SOAP bindings for protocols other than HTTP (e.g., FTP, vendor-specific messaging protocols).

As complex and hierarchical data types are widely used, you can also expect cross-vendor interoperability for hierarchical data shapes such as JDBC WebRowset and ADO Datasets, so as to allow data to be exchanged between Web services consumers and producers without requiring XML Schema Definition information. You can also expect a convergence between database Web services and (data-centric) Grid data services for service description and publication, service invocation, event subscription, and notification.

Conclusion

Database Web services leverage your existing server-side infrastructure, allowing you to use your database as both service provider and service consumer. I showed you how we at Oracle allow you to turn your database into a Web service provider, thus enabling you to share data and metadata across corporate intranets and access the database operations, e.g. triggers, through SOAP requests. I also explained how you can turn your database into a Web services consumer to access dynamic data. Finally, I gave you an overview of future database Web services capabilities.

A LIMITED TIME SAVINGS

SUBSCRIBE TODAY TO MULTI

Go To www.sys-con.com/suboffer.cfm

SYS-CON MEDIA
SAVE UP TO $400 FOR MULTIPLE SUBSCRIPTIONS
3-PACK • 6-PACK • 9-PACK – Limited Time Offer
3-Pack
Three-Pack Special Offer

and receive your **FREE** CD Gift Package VIA Priority Mail

Each CD is an invaluable developer resource packed with important articles and useful source code!

Pick the CDs to go with your Multi-Pack order

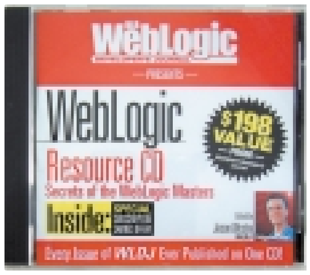
- **Pick one CD with your 3-Pack order**
- **Pick two CDs with your 6-Pack order**
- **Pick three CDs with your 9-Pack order**

☐ Web Services Resource CD
☐ Java Resource CD
☐ WebLogic Resource CD
☐ ColdFusion Resource CD
☐ XML Resource CD
☐ WebSphere Resource CD
☐ Linux Resource CD

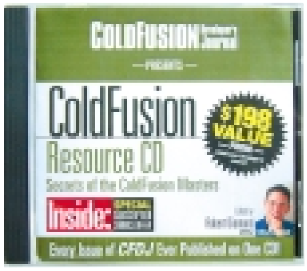
Your order will be processed the same day!

More than 1,400 Web services and Java articles on one CD! Edited by well-known editors-in-chief Sean Rhody and Alan Williamson, these articles are organized into more than 50 chapters on UDDI, distributed computing, e-business, applets, SOAP, and many other topics. Plus, editorials, interviews, tips and techniques!
LIST PRICE $198

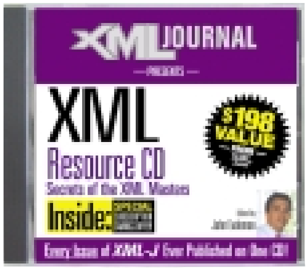
The most complete library of exclusive *JDJ* articles compiled on one CD! Assembled by *JDJ* Editor-in-Chief Alan Williamson, more than 1,400 exclusive articles are organized into over 50 chapters, including fundamentals, applets, advanced Java topics, Swing, security, wireless Java,... and much more!
LIST PRICE $198



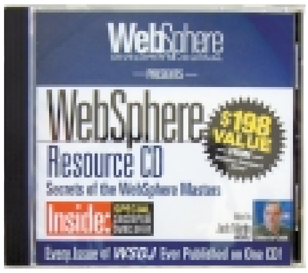
The most complete library of exclusive *WLDJ* articles ever assembled! More than 200 articles provide invaluable information on "everything WebLogic", including WebLogic Server, WebLogic Portal, WebLogic Platform, WebLogic Workshop, Web services, security, migration, integration, performance, training...
LIST PRICE $198



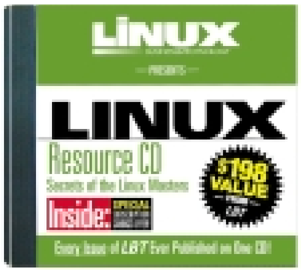
The most complete library of exclusive *CFDJ* articles on one CD! This CD, edited by *CFDJ* Editor-in-Chief Robert Diamond, is organized into more than 30 chapters with more than 400 exclusive articles on CF applications, custom tags, database, e-commerce, Spectra, enterprise CF, error handling, WDDX... and more!
LIST PRICE $198



The largest and most complete library of exclusive *XML-Journal* articles compiled on one CD! Edited by well-known Editors-in-Chief Ajit Sagar and John Evdemon, these articles are organized into more than 30 chapters containing more than 1,150 articles on Java & XML, XML & XSLT, <e-BizML>, data transition... and more!
LIST PRICE $198



The most up-to-date collection of exclusive *WSDJ* articles! More than 200 articles offer insights into all areas of WebSphere, including Portal, components, integration, tools, hardware, management, sites, wireless programming, best practices, migration...
LIST PRICE $198



An up-to-the-minute collection of exclusive *Linux Business & Technology* articles plus Maureen O'Gara's weekly *LinuxGram*, which keeps a finger on the pulse of the Linux business community. Edited by *LBT's* Editor-in-Chief Alan Williamson, these articles cover migration, hardware, certification, and the latest Linux-related products. Available June 2003!
LIST PRICE $198

Subscribe Online Today

PLE MAGAZINES ONLINE

AND SAVE UP TO $400 AND RECEIVE UP TO 3 FREE CDs!

RECEIVE YOUR DIGITAL EDITION ACCESS CODE INSTANTLY WITH YOUR PAID SUBSCRIPTIONS

TO ORDER •Choose the Multi-Pack you want to order by checking next to it below. •Check the number of years you want to order. •Indicate your location by checking either U.S., Canada/Mexico or International. •Then choose which magazines you want to include with your Multi-Pack order.

Pick a 3-Pack, a 6-Pack or a 9-Pack

☐ 3-Pack	☐ 1YR	☐ 2YR	☐ U.S.	☐ Can/Mex	☐ Intl.
☐ 6-Pack	☐ 1YR	☐ 2YR	☐ U.S.	☐ Can/Mex	☐ Intl.
☐ 9-Pack	☐ 1YR	☐ 2YR	☐ U.S.	☐ Can/Mex	☐ Intl.

☐ **Linux Business & Technology**

U.S. - Two Years (24) Cover: $143	You Pay: $79.99 /	Save: $63 + FREE $198 CD
U.S. - One Year (12) Cover: $72	You Pay: $39.99 /	Save: $32
Can/Mex - Two Years (24) $168	You Pay: $119.99 /	Save: $48 + FREE $198 CD
Can/Mex - One Year (12) $84	You Pay: $79.99 /	Save: $4
Int'l - Two Years (24) $216	You Pay: $176 /	Save: $40 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **Java Developer's Journal**

U.S. - Two Years (24) Cover: $144	You Pay: $89 /	Save: $55 + FREE $198 CD
U.S. - One Year (12) Cover: $72	You Pay: $49.99 /	Save: $22
Can/Mex - Two Years (24) $168	You Pay: $119.99 /	Save: $48 + FREE $198 CD
Can/Mex - One Year (12) $84	You Pay: $79.99 /	Save: $4
Int'l - Two Years (24) $216	You Pay: $176 /	Save: $40 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **Web Services Journal**

U.S.- Two Years (24) Cover: $168	You Pay: $99.99 /	Save: $68 + FREE $198 CD
U.S. - One Year (12) Cover: $84	You Pay: $69.99 /	Save: $14
Can/Mex - Two Years (24) $192	You Pay: $129 /	Save: $63 + FREE $198 CD
Can/Mex - One Year (12) $96	You Pay: $89.99 /	Save: $6
Int'l - Two Years (24) $216	You Pay: $170 /	Save: $46 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **.NET Developer's Journal**

U.S. - Two Years (24) Cover: $168	You Pay: $99.99 /	Save: $68 + FREE $198 CD
U.S. - One Year (12) Cover: $84	You Pay: $69.99 /	Save: $14
Can/Mex - Two Years (24) $192	You Pay: $129 /	Save: $63 + FREE $198 CD
Can/Mex - One Year (12) $96	You Pay: $89.99 /	Save: $6
Int'l - Two Years (24) $216	You Pay: $170 /	Save: $46 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **XML-Journal**

U.S. - Two Years (24) Cover: $168	You Pay: $99.99 /	Save: $68 + FREE $198 CD
U.S. - One Year (12) Cover: $84	You Pay: $69.99 /	Save: $14
Can/Mex - Two Years (24) $192	You Pay: $129 /	Save: $63 + FREE $198 CD
Can/Mex - One Year (12) $96	You Pay: $89.99 /	Save: $6
Int'l - Two Years (24) $216	You Pay: $170 /	Save: $46 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **WebLogic Developer's Journal**

U.S. - Two Years (24) Cover: $360	You Pay: $169.99 /	Save: $190 + FREE $198 CD
U.S. - One Year (12) Cover: $180	You Pay: $149 /	Save: $31
Can/Mex - Two Years (24) $360	You Pay: $179.99 /	Save: $180 + FREE $198 CD
Can/Mex - One Year (12) $180	You Pay: $169 /	Save: $11
Int'l - Two Years (24) $360	You Pay: $189.99 /	Save: $170 + FREE $198 CD
Int'l - One Year (12) $180	You Pay: $179 /	Save: $1

☐ **ColdFusion Developer's Journal**

U.S. - Two Years (24) Cover: $216	You Pay: $129 /	Save: $87 + FREE $198 CD
U.S. - One Year (12) Cover: $108	You Pay: $89.99 /	Save: $18
Can/Mex - Two Years (24) $240	You Pay: $159.99 /	Save: $80 + FREE $198 CD
Can/Mex - One Year (12) $120	You Pay: $99.99 /	Save: $20
Int'l - Two Years (24) $264	You Pay: $189 /	Save: $75 + FREE $198 CD
Int'l - One Year (12) $132	You Pay: $129.99 /	Save: $2

☐ **Wireless Business & Technology**

U.S. - Two Years (24) Cover: $144	You Pay: $89 /	Save: $55 + FREE $198 CD
U.S. - One Year (12) Cover: $72	You Pay: $49.99 /	Save: $22
Can/Mex - Two Years (24) $192	You Pay: $139 /	Save: $53 + FREE $198 CD
Can/Mex - One Year (12) $96	You Pay: $79.99 /	Save: $16
Int'l - Two Years (24) $216	You Pay: $170 /	Save: $46 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **WebSphere Developer's Journal**

U.S. - Two Years (24) Cover: $360	You Pay: $169.99 /	Save: $190 + FREE $198 CD
U.S. - One Year (12) Cover: $180	You Pay: $149 /	Save: $31
Can/Mex - Two Years (24) $360	You Pay: $179.99 /	Save: $180 + FREE $198 CD
Can/Mex - One Year (12) $180	You Pay: $169 /	Save: $11
Int'l - Two Years (24) $360	You Pay: $189.99 /	Save: $170 + FREE $198 CD
Int'l - One Year (12) $180	You Pay: $179 /	Save: $1

☐ **PowerBuilder Developer's Journal**

U.S. - Two Years (24) Cover: $360	You Pay: $169.99 /	Save: $190 + FREE $198 CD
U.S. - One Year (12) Cover: $180	You Pay: $149 /	Save: $31
Can/Mex - Two Years (24) $360	You Pay: $179.99 /	Save: $180 + FREE $198 CD
Can/Mex - One Year (12) $180	You Pay: $169 /	Save: $11
Int'l - Two Years (24) $360	You Pay: $189.99 /	Save: $170 + FREE $198 CD
Int'l - One Year (12) $180	You Pay: $179 /	Save: $1

3-Pack

Pick any 3 of our magazines and save up to $275.00

Pay only $175 for a 1 year subscription plus a FREE CD

- 2 Year – $299.00
- Canada/Mexico – $245.00
- International – $315.00

6-Pack

Pick any 6 of our magazines and save up to $350.00

Pay only $395 for a 1 year subscription plus 2 FREE CDs

- 2 Year – $669.00
- Canada/Mexico – $555.00
- International – $710.00

9-Pack

Pick 9 of our magazines and save up to $400.00

Pay only $495 for a 1 year subscription plus 3 FREE CDs

- 2 Year – $839.00
- Canada/Mexico – $695.00
- International – $890.00

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

www.sys-con.com/2001/sub.cfm

SYS-CON MEDIA

Reviewed by Paul Kaiser

Grand Central Communications' Web Services Network *Going down the right track*

Author Bio:
Paul Kaiser is a consultant for Info Technologies, Inc., in New Jersey, where he focuses on application integration.
PAULKAISER@YAHOO.COM

Integration efforts within an enterprise have been aided by the adoption of service-oriented architectures and common integration infrastructure. While the service-oriented architecture needs to be driven from within the organizations, a common infrastructure can come from outside. Grand Central Communications offers this via its Web Services Network.

The Web Services Network

The Web Services Network provides, as a service, a framework that lets companies integrate their existing applications in a flexible, secure, extensible manner. Think of it as an integration broker available in an ASP model. It offers multiple ways to get connected even if you're not Web services enabled. But for those who are, SOAP is its native tongue.

Mediation

The network helps mediate differences in technologies used by different organizations. While SOAP is its native protocol, the network supports FTP, SMTP, and non-SOAP HTTP among others. It also provides support for data transformations and message routing. The mediation isn't limited to protocol and data format, but supports both synchronous and asynchronous messaging modes. This flexibility reduces the coupling between you and your partners. For many organizations, there is nothing about the network that would require firewall configuration changes.

The network supports messaging using both request-response and notification (one-way) patterns. Persistence, guaranteed delivery, and message expiration are available for asynchronous messages to provide the reliability required for business transactions. And both push- and pull-based delivery is supported.

End-to-End Accountability

As an independent service provider Grand Central can establish itself as a trusted third party to the transactions that flow through its network. In this role, it provides visibility into the status of any interaction. A message moving from one service to another (a call) generates multiple status entries that can be viewed by the service owners. Each call can be part of a larger exchange (a session). When a message is initiated on the network, the initiator can choose to include it in an existing session or create a new session.

Alerts can be set to generate notifications for exceptional events at the network level. Partners can even configure routing rules to produce customized event-handling mechanisms.

Extensibility

Providing a framework for service extensions is an important aspect of any platform purporting flexibility. Grand Central leverages Web services as the standard for plugging in new features. The services may be a standard network service; a company's service; a partner's service; or a third-party service, such as an ASP. The network supports routing services to chain multiple services together and a transformation service to translate the data to the appropriate form.

Security and Trust

Grand Central establishes a common concept of identity and trust on their network. Each service within the organization is assigned an identity and may authenticate in one of three ways: username/password over HTTP, username/password over HTTPS, or certificate-based authentication over HTTPS. These three forms are used to define the levels of both security and security policies. The security level associated with your service specifies the security protocol between your service implementation and the network. The security policy specifies the minimum security level required for other services on the network to connect to you.

Access control to your service is available at both an implicit and explicit level. You can grant or deny access from other services as a default. And, you can explicitly grant or deny access from specific services.

The organizational identity and detailed message status information form the support for nonrepudiation support in the network. Every participant knows who invoked the service, when, and what the outcome was.

GRAND CENTRAL COMMUNICATIONS

COMPANY INFO
Grand Central Communications, Inc.
50 Fremont Street, 16th Floor
San Francisco, California 94105
415.344.3200

EVALUATION DOWNLOAD
Evaluation accounts are available on their developer Web site (http://developer.grandcentral.com/register/index_html).

PRICING
Per-partner per-month basis
Sites
www.grandcentral.com (Corporate)
http://developer.grandcentral.com (Developer)

Web Services Journal MAGAZINE WSJ World Class PRODUCT AWARD

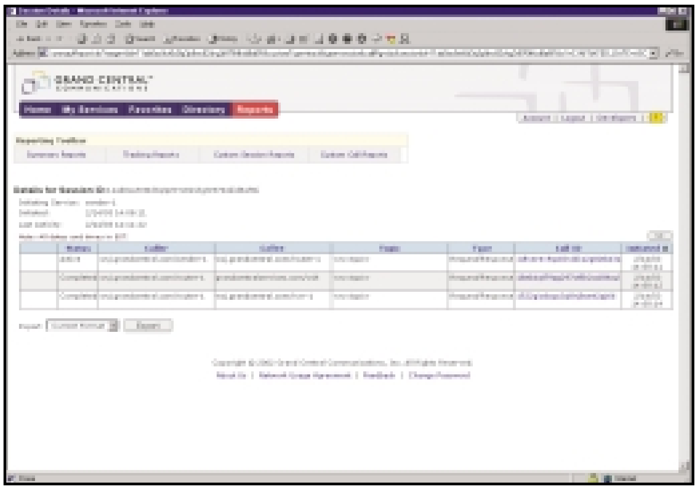


FIGURE 1 Session details report

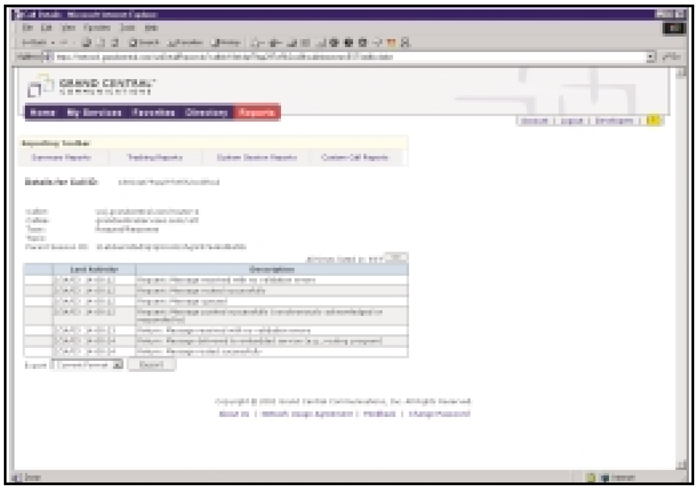


FIGURE 2 Call details report

A Simple Start

Getting started is simple. Even if you don't already have a Web services infrastructure in place, you can take a test drive of what it has to offer. The developer site has enough documentation and examples to get you going.

Setting up a simple routing and transformation sample was easy. With a couple of endpoint services and a routing service, you get an idea of what the network offers. Use a Grand Central sub-domain for your evaluation account. Create your services from the MyServices page of your network account.

Endpoint services are associated with message queues and let you send and receive messages on the network. For now, give the endpoint services meaningful names and unique addresses and use the defaults elsewhere. Create two endpoint services: one to send messages and one to receive them.

Routing services let you define rules and service invocation sequences to create a simple sequence of if-then constructs. The first rule whose logical condition evaluates as true is executed. Each rule may sequence one or more services to form a pipe and filter style process.

Rule expressiveness is limited: only the sender address and message topic can be evaluated, only the equality operator is available, and you can only specify single values or leave blank to match all values. Leave the sender and topic criteria blank. Next, specify the services to which the message will be routed.

Add the transformation service as the first action. Supply the address for the XML Transformation service. The Address Book feature is very convenient here. Use the defaults for message type (original) and topic (current). Specify the body:

```
<m:transform
 xmlns:m="http://www.grandcentralser-
vices.com/
  schemas/Transformation">
 <input href="cid:people.xml"/>
 <xsl href="file:people.xsl"/>
 <output href="cid:people_new.xml"/>
</m:transform>
```

Note that the href attribute of <input> and <output> tags uses the "cid:" prefix to refer to a message attachment and the <xsl> tag's href attribute uses the "file:" prefix to refer to the stylesheet. The stylesheet was uploaded to Grand Central's Document Manager service (www.grandcentralservices.com/docman), a simple facility for managing static files.

Next we add a second action to the routing rule that sends the response from the transformation service to the receiver service created above. That completes the routing service.

You can send a message to your routing service using a SOAP client, HTTP client, or Grand Central's Web messaging user interface. For brevity, I'll describe the user interface. From the MyServices page, select the Send service. Select Compose New to create a new message. Put the routing service address in the "To" field. A topic isn't required. Attach a file with the name as referenced in the <input> tag's href attribute. Send the message. If all goes well, a postmark response will indicate the new session ID and the receiver service will have a message waiting in its queue. If an error occurs, the fault response will indicate the session and call identifiers. Figures 1 and 2 show the session and call details.

Conclusion

Grand Central is in a unique position to simplify integration between organizations. The network reduces coupling by mediating protocol, format, and messaging mode. They are positioned as a neutral third party and provide good visibility to message status details. The network runs 24x7 and offers standard service-level agreements for delivery, security, and operation. Grand Central can also tailor the SLA to your requirements.

If you need the robust workflow and routing of off-the-shelf integration brokers, you may be disappointed. Later this year, Grand Central plans to provide the services and infrastructure to enable a loosely coupled, services-oriented architecture including mediation support for WS-I standards, support of advanced loosely coupled routing, and long-lived transactions. While routing services may be nested to form deep decision trees, rule selection and content access need improvement. Sending a message down multiple parallel paths has to occur outside the network and correlating the paths may prove challenging under certain configurations.

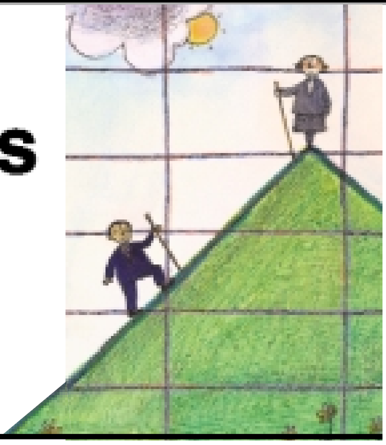
But don't be discouraged. Grand Central is on the right track in support of external partner integration. Reducing interface coupling is very important to long-lived implementations. Since you have to concern only yourself with how your applications connect, the network makes it easier to scale the number of external partner integrations beyond the top tier.

So take a train to the station… you might just like the ride.

written by Chandra Venkatapathy and David Melgar

Message-Centric Web Services vs. RPC-Style Invocations

When SOAP is a preferred method



The notion of distributed computing has been evolving for a long time, during which we have been building business solutions by integrating various systems and platforms. Typically, these interactions are characterized by accessing and invoking clearly defined interfaces using well-known communication mechanisms like Remote Procedure Calls (RPCs) or by using message-centric protocols. Web services is probably the easiest distributed computing paradigm available today. Given the familiarity to RPC and message-centric invocation models, we need the flexibility of using both these models in the Web services invocation frameworks.

AUTHOR BIO:

David Melgar has a background in the retail industry, systems management, Java technology, and Web services. The original author of UDDI4J, David is a member of the Emerging Technologies division of IBM software group, and focuses on Web services security within the IBM Web Services Toolkit.

DMELGAR@US.IBM.COM

Chandra Venkatapathy is a market manager for WebSphere Business Integration in the IBM software group and has been working in Web services and business integration.

CHANDRAV@US.IBM.COM

Luckily, Web services use a rather simple framework for communication – Simple Object Access Protocol (SOAP). With SOAP, you can use both message-centric invocation and RPC to invoke Web services. In this article we'll discuss the evolution of RPC and message-centric invocations, and the pros and cons of their programming models in a service-oriented architecture.

Introduction to Web Services

Web services is nothing but a set of open standards for distributed computing and is being widely adopted by the industry. Using Web services, you can describe an existing business process as an easily consumable interface, search for other business processes as Web services components inside and outside of the enterprise, and invoke them as needed. Processes that are limited by architecture-, platform-, and language-specific implementations can be expressed as open standards–based Web services components. Using these components as building blocks, you can build robust business applications dynamically without worrying about platform, system, or enterprise boundaries, thus creating a true service-oriented architecture.

The basic invocation framework for Web services is based on SOAP, a simple, lightweight, transport-agnostic protocol that is based on XML. SOAP has evolved from RPC and messaging principles and provides support for both types of invocations. Current business applications rely on their integration infrastructure to provide robust delivery of messages and transactional integrity. Similarly, Web services–based business applications require the same messaging and transactional attributes as current applications. As SOAP is the invocation framework for Web services, we'll discuss how these requirements are addressed later.

In order to fully understand the evolution of SOAP, it's important to first understand the evolution of RPC and messaging systems.

Evolution of RPC

The notion of procedure calls has been in use for a long time. In the programming languages of yesteryear, we used procedure calls for programming simplicity. To kindle our nostalgia, we created modules of structured programming code and defined them as (local) procedures. We invoked these procedures with clearly defined parameters. Soon Remote Procedure Calls evolved as applications were distributed over the network. The RPC framework helped us cross the boundary of the local machine to con-

JAVA: INDUSTRY NEWSLETTER
Reply
Reply All
Forward
Flag
Print
Email Newsletters
JAVA DEVELOPERS JOURNAL Industry > Newsletter

BROUGHT TO YOU BY...

aligo Wireless Development Strategies - A Free White Paper. How can you use J2EE/J2ME to successfully deploy Wireless Applications? Aligo, the leader in Java based wireless application servers, offers a free technical paper on Java and Wireless Development. Click Here! for more information.

JAVA DUDES

JDJ NEWS DESK EXCLUSIVE

Oracle Targets Top Spot With New Oracle9iAS and JDeveloper

by Rick Ross - Founder of Javalobby.org

(December 13, 2001) – I returned just yesterday from a fantastic trip to San Francisco for Oracle's OpenWorld conference. It wasn't clear what to expect when I accepted Oracle's invitation to give a presentation, but I was delighted to find the show had a high-energy crowd that was very tuned in to Java, especially J2EE and Web services. The whole show was great, but the most interesting and important items came from Oracle, itself - Oracle9i Application Server Release 2 and Oracle9i JDeveloper.

Newsletter

JDJ News
JDJ List
Subscribe Now!
Special Offer!
Privacy Notice

WAKESOFT Learn more now!

Introducing AltoWeb Release 2.5

WEB SERVICES: INDUSTRY NEWSLETTER
Reply
Reply All
Forward
Flag
Print
Email Newsletters
WebServices JOURNAL Industry >Newsletter

Stay Tuned for WSJ's December Issue!
Full Editorial Content Coming December 5th

BROUGHT TO YOU BY...

ORACLE Catch the Oracle9i JDeveloper wave and see how built-in profilers and Code Coach makes your Java code tighter and faster than ever before. Based on the latest J2EE and XML standards, JDeveloper combines Java coding, UML modeling, and RAD on NT/Linux/Unix. Click Here! for more information.

ONLINE EXCLUSIVE: FULL "KEYNOTE PANEL" TRANSCRIPT

[illegible]

Sean Rhody: Probably the question that is on almost everybody's mind is, what are Web services? I know I have a whole magazine devoted to answering that question, but I think one of the most interesting parts of the magazine is that the answers are different for different people and from different people. So to start things off I'll go down and ask each of the panelists that question. In your mind, Barry, what are Web services?

Barry Morris: I would think we all tend

Newsletter

WSJ News
Next Issue
Subscribe Now!
Privacy Notice

spiritsoft complete integration

BROUGHT TO YOU BY...

flamenco Secure Web services. Reliable Web services. Manageable Web services. If you need Web services then you need a Web services network. Security, guaranteed delivery, and management services with no code changes. Plug into the Flamenco Network and get going! For a limited time, bring your Web services and test-drive the Flamenco Network for free.

[illegible]

XML NEWS

Xerces-C++ XML Parser Version 1.6.0 Provides Full Support for W3C XML Schema Recommendation

(Toronto) - The XML Parsers Development division of the IBM Toronto Laboratory has announced the much-anticipated release of the Apache Xerces-C++ XML parser version 1.6.0, including full support for the W3C XML Schema Recommendation version 1.0. Xerces-C++ is a validating XML parser written in a portable subset of C++; Xerces-C++ makes it easy to give applications the ability to read and write XML data. A shared library is provided for parsing, generating, manipulating, and validating XML documents. [illegible] features: conformance to the XML Specification 1.0; tracking of [illegible] [Level 1.0], DOM [Level 2.0], SAX/SAX2, namespace, and XML [illegible] source code, samples, and documentation; [illegible] and validation of XML; pluggable catalogs, validators [illegible] performance; and customizable error handling. The latest [illegible] C++ includes a port to FreeBSD and allows users to [illegible] through a method call. Source code and binaries [illegible] Linux, Solaris, and Windows.

[illegible] high performance, modularity, and scalability. Source [illegible] are provided with the parser. For [illegible] to make minimal use of templates, no RTTI [illegible] minimal use of #ifdefs.

Newsletter

XML News
Subscribe Now!
Special Offer!
Privacy Notice

THE INSIDER INTELLIGENCE YOU NEED...

TO KEEP AHEAD OF THE CURVE

FREE E-Newsletters

SIGN UP TODAY!

Go to www.SYS-CON.com

The most innovative products, new releases, interviews, industry developments, and plenty of solid *i*-technology news can be found in SYS-CON Media's Industry Newsletters. Targeted to meet your professional needs, each e-mail is informative, insightful, and to the point. They're free, and your subscription is just a mouse-click away at www.sys-con.com.

SELECT THE INDUSTRY NEWSLETTERS THAT MATCH YOUR NEEDS!
CHOOSE ONE – OR TRY THEM ALL!

JAVA DEVELOPERS JOURNAL Industry > Newsletter	WebLogic Industry > Newsletter
WebServices Industry >Newsletter	WebSphere Industry > Newsletter
XML JOURNAL Industry > Newsletter	COLDFUSION Journal Industry > Newsletter
wireless Industry > Newsletter	.NET Journal Industry > Newsletter

Don't Delay! Subscribe for FREE! at www.sys-con.com

Exclusively from the World's Leading *i*-Technology Publisher SYS-CON MEDIA nect to another machine to invoke a specific module with a set of parameters. Since RPCs are parametric invocations, the RPC framework has to handle the complexity of reconciling the system and language boundaries – often called "marshalling" – between the caller and the called.

In the late '90s, XML-RPC was proposed, in which XML was used to marshal data. Given the widespread support for XML, XML-RPC offered a simplistic, yet viable invocation model over the Internet using HTTP. However, use of XML-RPC was limited as SOAP evolved rapidly and offered richer semantics than XML-RPC. Since SOAP is built on XML, you can use it to make RPC invocations. RPCs are typically used in the synchronous programming model. That is, the caller invokes a RPC on another machine, and waits for the RPC to return the results.

Evolution of Message-Centric Architecture

In a message-centric architecture, applications connect to other applications using well-defined message sets through message-oriented middleware (MOM). Initially, message-centric architecture was used to connect applications running on mainframes. As many early applications were performing batch processing, there was a need for queuing requests. Early messaging middleware offered queuing for asynchronous processing, persistence of the messages, and assured delivery in addition to connectivity. The strength of messaging middleware is determined not only by its support for the above-mentioned features, but also by support for a wide variety of platforms, languages, transport protocols, and open standards. A message-centric programming model, although used for both synchronous and asynchronous invocations, is widely known for asynchronous invocation.

Differences Between Message-Centric Invocations and RPC

The main difference between message-centric invocation and RPC lies in the programming model. RPC is a good model for function-centric invocations. As discussed earlier, it works well when we have clearly defined functions and associated parameters. The parameters can be simple data types or complex objects, in which case the objects have to be serialized as bits and bytes when traversing through the wire, often resulting in performance degradation. RPCs are typically associated with synchronous functional invocations and do not support messaging semantics such as assured delivery.

The message-centric model is frequently used for data-centric programming. That is, data is exchanged in a prescribed message format that both the sender and receiver can understand. Hence there is an emphasis on data delivery using native communication protocols and programming support to send and receive data. Often, these programming models are used for loosely coupled integration with messages and events flowing back and forth. The strength of the message-centric model lies primarily in the implementation of the messaging system, such as support for a wide variety of platforms and languages, assured delivery, persistence, support for asynchronous invocations, support for open standards, and performance. Table 1 captures the main differences.

Current SOAP engines provide interfaces for both messaging and RPC. Use of one methodology does not preclude use of another, even within the same application.

TABLE 1: Difference Between RPC and Message-Centric Invocations

RPC Invocation	Message-Centric Invocation
Function-centric: Aligns well when service invocation is thought of as a procedural call.	Data-centric: Aligns well when there is an exchange of data between applications.
Works well for data and objects that can be serialized easily.	Data need not have an object representation. Data can be sent as a payload.
Synchronous invocation: Applications make a request and wait for the response.	Asynchronous invocation: Applications make a request and don't wait for any response. Instead, they rely on events notifications.
	Strong notion of message semantics like assured delivery.

Web Services and Message-Centric Invocations

In the Web services world, messaging middleware plays an important role. The SOAP framework offers a platform-and transport-agnostic protocol to invoke Web services. However, it does not address many of the messaging semantics such as assured delivery, support for asynchronous messages, and persistence. SOAP messages can be bound to any transport, such as HTTP for example. However, the typical invocation of Web services is implemented by sending SOAP using HTTP. The problem is that HTTP does not offer assured delivery nor does it have asynchronous capabilities. If your application demands message semantics such as robust delivery of Web services and asynchronous messaging capability, you should bind SOAP-based Web services to robust messaging middleware like IBM's WebSphere MQ.

SOAP provides the header semantics that allow intermediaries to provide higher-level functions to the message flow. Protocols have been discussed which address reliable delivery, transaction processing, correlation, routing, security, and more.. When these protocols are used as part of the SOAP messaging flow, their benefits can be achieved without changing the semantics used to invoke the Web service. In other words, these benefits can be achieved regardless of RPC or message-based invocation style.

Now that we've discussed the evolution of RPC and message-centric invocations, let's look at how SOAP addresses both invocation models.

Evolution of SOAP: How You Can Do Both

SOAP is intended to be a simple yet powerful extensible protocol for distributed messaging based on XML. A key element of SOAP is its concept of having a message envelope and a set of headers. The envelope contains the message body that can be XML data or parameters and is intended for the recipient. As a SOAP message can hop through several intermediaries before it reaches its destination, the header contains routing and message-handling semantics for the intermediaries.

Another key extensibility point is that SOAP can be bound to various transport protocols. HTTP is the default binding specified within the SOAP specification and its use has been one of the key contributors to the success of SOAP. Although HTTP is the most common transport, it is by no means the only one. SOAP is designed such that its semantics are largely independent of the transport binding. This allows a client to remain unaffected if alternate transports are used. Other transports can be used to provide reliable delivery or other capabilities. Other bindings could include JMS (Java Message Service) and WebSphere MQ.

SOAP's capabilities have provided for the transport and data format neutrality necessary for successful distributed computing. Previous distributed models such as RPC and CORBA specified more complex common data formats, but did not provide a simple and replaceable transport binding. SOAP is poised to satisfy the market need for a platform-agnostic distributed computing model.

The SOAP model has encouraged the development of follow-up specifications to enable higher-level functions. These functions include:

- ***Security:*** SOAP's support for intermediaries requires that messages provide a means of end-to-end security, beyond the traditional transport-level security offered by SSL. The WS-Security lays the groundwork for core security constructs.
- ***WS-Coordination, WS-Transaction, and BPEL4WS (Business Process Execution Language for Web Services)*** provide transaction and business flow processing.These specifications are available from www-106.ibm.com/developerworks/webservices/library/ws-spec.html.

Invocation Model

Most SOAP implementations support both RPC and messaging interfaces. As an illustration of the differences, the following method signatures illustrate client side APIs using the Java SOAP engine Axis.

RPC-Style Invocation

Let's take an example where a procurement application gets a quote for a purchase from a supplier. In the example discussed below, the "GetPurchaseInfo" service has a method called 'getQuote "ForPart" with a clearly defined parameter called "PartNumber". Typically, the underlying SOAP implementation creates a client stub for invocation that can easily be invoked from the financial application. The invocation looks just like a local method invocation as shown below, yet would invoke the method on the remote Web service:

```
Float f =
GetPurchageInfoStub.getQuoteForPart(String
PartNumber)
```

Web services provide the flexibility of dynamically determining the service signature. In this case, a stub has not been generated, and hence dynamic invocation is used. The client dynamically determines the method to call and parameters to invoke. This is conceptually similar using introspection to invoke a Java method.

"Web services is probably the easiest distributed computing paradigm available today"

```
Service service = new Service(new
URL(wsdl_url), serviceQN );
Call call = (Call) service.createCall(
portQN, "getQuoteForPart" );
Float flt = (Float) call.invoke( new
Object[] { "Cartridge" } );
```

In either case, the actual SOAP message is similar to the following:

```
<soapenv:Envelope>
 <soapenv:Body>
  <ns1:getQuoteForPart>
   <symbolxsi:type="xsd:string">XXX</
      symbol>
  </ns1:getQuoteForPart>
 </soapenv:Body>
</soapenv:Envelope>
```

Message-Centric Invocation

In a message-centric invocation, the message content is XML obtained elsewhere, such as a database. In the example, the same procurement application sends a purchase order to the supplier. A purchase order is usually a long document with various data elements well suited to representation in XML. If GetPurchaseOrder is implemented as a service, it's easier to send the whole purchase order document rather than send each and every field as a parameter.

In the following example, a SOAP message is constructed with Purchase Order XML document and sent to the supplier using a message-style invocation.

```
SOAPBodyElement body = new
SOAPBodyElement( purchase order XML
document-- );
call.invokeOneWay( params );
```

The result means nothing if the request doesn't have a reply, or another SOAPBody Element from which the original XML is obtained. As we noted earlier, if the business case requires assured delivery of the purchase order, the SOAP message should be sent over a transport like WebSphere MQ.

Multiple-Message Invocation

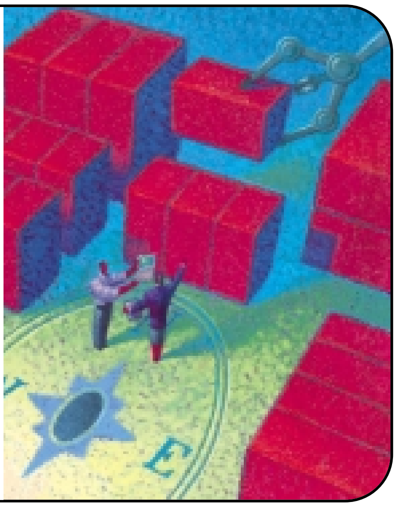
Business activities are often associated with integrating services from distributed applications and are often associated with the notion of transaction. The transactional integrity is maintained by transactional systems that implement two-phase protocols. We have similar support for Web services with the announcement of specifications like WS-Coordination, WS-Transaction, and BPEL4WS.

Conclusion

SOAP provides rich semantics for RPC and message-centric invocations. As Web services are manifestations of business-critical processes, invocation of these services needs to have the same robust message and transaction semantics regardless of the style of invocation. SOAP provides the flexibility to bind with the transport of your choice and therefore can enable robust message delivery. There are ongoing efforts to provide a specification for the reliable delivery of Web services; in the meantime, several commercial implementations already support assured delivery of Web services.

Web Services and Portal Technology

Deliver services – anytime, anywhere, anyhow



Portals are central points of access for applications and content for both internal and external use in an enterprise through interactive and rich presentation interfaces.

Personalized access to information, applications, processes, and people is provided to portals by getting information from local or remote data sources such as databases, transaction systems, content providers, or remote Web sites. This information is then rendered and aggregated into meaningful pages to provide information to users in a compact and easily consumable form. In addition to pure information, many portals also include applications like e-mail, calendars, organizers, banking, bill presentation, host integration, and many more.

Different layout, profile, content, and access criteria are the cornerstones for the display and selection options that shape the information that can be accessed through a portal. Today portals are made up of building blocks – portlets – that plug into a portal's base infrastructure and enable the aggregation of interactive rendering markup. A personalized Web page in a portal can be made up of numerous portlets that provide specific functionality and expose various external services and content.

The lack of standards has led portal server platform vendors to define proprietary APIs for local portal components and for invocation of remote components that create interoperability problems for portal customers, application vendors, content providers, and portal software vendors. The Java Portlet API and Web Services for Remote Portals (WSRP) standards are being developed to overcome these problems, providing interoperability between portlets and portals, and between portals and user-facing Web services.

Let's consider that a Road Warrior Portal manager for a sales organization wants to include a Travel Expense service to calculate travel expenses, and an external Map service to provide useful location-based information.

To add more functionality and services to a portal, you need to add more portlets to render and display the new features, leading to increased cost and time investment. It would be much more convenient if the Travel Expense Authorization and Map Web services included both business and presentation logic.

WSRP are interactive Web services that can be called through an interface built into the portal itself – a proxy. This eliminates both the need for code on the portal as well as the redeployment of presentation details each time (see Figure 1).

Web services can enable the seamless and hassle-free integration of various applications and content into the portal software through cost-effective integration and adherence to open standards. Web services are software applications and functionality that can be delivered over the Web. Portlets, the individual components delivered by portals, can be treated as Web services.

How do we ensure that Web services can deliver custom marked up content to various devices through a portal, and secondly, how do we control access to Web services and other portal content. The two technologies available for handling these complexities are the Open Solution and J2EE.

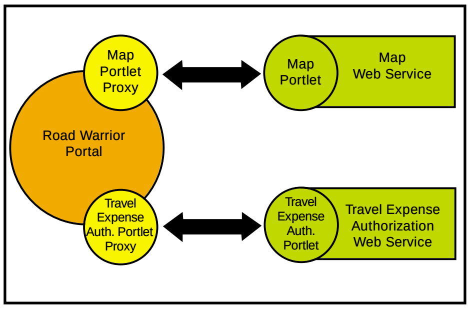

Map Portlet Proxy
Road Warrior Portal
Travel Expense Auth. Portlet Proxy
Map Portlet
Map Web Service
Travel Expense Auth. Portlet
Travel Expense Authorization Web Service

FIGURE 1 Travel Expense Portlet and Map Portlet that interact with smart Web services

Delivering Custom Content

When you compare Web services and portal technology you see that they have a lot in common. At the heart of Web services is an obvious dependence on HTTP and XML, which are also the building blocks of Web technology, and hence portal infrastructures.

At this point you should understand the difference between the processing capabilities of mobile devices and Web services clients. Mobile clients can typically receive and consume some form of markup that the device is built to handle, over HTTP or a wireless protocol. Web services

Author Bios

Ash Parikh has over 10 years of computer and IT experience, including object-oriented analysis and design, distributed architecture, middleware architecture, and software design and development. He has served in many architect-level roles with technology leaders such as Oracle Corporation, BEA Systems, Sun Microsystems, and PeopleSoft. Ash is the president of the Bay Area Chapter of the Worldwide Institute of Software Architects, an initiative close to his heart, through which he evangelizes various software architecture paradigms. He is a contributing author of Oracle9iAS Building J2EE Applications (Osborne Press). ASHISHPA@HOTMAIL.COM

Ashwin Rao, the vice president of the Bay Area Chapter of the Worldwide Institute of Software Architects, has over six years of experience in software development, having begun his career writing defense industry software for real-time, embedded systems. His current focus is J2EE architecture and development and Web services. Ashwin has authored a number of articles on Web services. ARAO@MAIL.COM

Premiering June 2003 at JavaOne

Sun's 2003 Worldwide Java Developer Conference

www.sys-con.com

Millions of Linux Users One Magazine

Linux Business and Technology

There is no escaping the penetration of Linux into the corporate world. Traditional models are being turned on their head as the open-for-everyone Linux bandwagon rolls forward.

Linux is an operating system that is traditionally held in the highest esteem by the hardcore or geek developers of the world. With its roots firmly seeded in the open-source model, Linux is very much born from the "if it's broke, then fix it yourself" attitude.

Major corporations including IBM, Oracle, Sun, and Dell have all committed significant resources and money to ensure their strategy for the future involves Linux. Linux has arrived at the boardroom.

Yet until now, no title has existed that explicitly addresses this new hunger for information from the corporate arena. *Linux Business & Technology* is aimed squarely at providing this group with the knowledge and background that will allow them to make decisions to utilize the Linux operating system.

Look for all the strategic information required to better inform the community on how powerful an alternative Linux can be. *Linux Business & Technology* will not feature low-level code snippets but will focus instead on the higher logistical level, providing advice on hardware, to software, through to the recruiting of trained personnel required to successfully deploy a Linux-based solution. Each month will see a different focus, allowing a detailed analysis of all the components that make up the greater Linux landscape.

Regular features will include:

Advice on Linux Infrastructure
Detailed Software Reviews
Migration Advice
Hardware Advice
CEO Guest Editorials
Recruiting/Certification Advice
Latest News That Matters
Case Studies

SAVE 30% OFF!

REGULAR ANNUAL COVER PRICE $71.76

YOU PAY ONLY $49.99

12 ISSUES/YR

*OFFER SUBJECT TO CHANGE WITHOUT NOTICE

SUBSCRIBE TODAY!
WWW.SYS-CON.COM
OR CALL
1-888-303-5282

LINUX EDGE conference & expo

June 3–5	LONDON
June 24–26	BERLIN
September	HONG KONG
October	CALIFORNIA

FOR ADVERTISING INFORMATION:
CALL 201 802.3020 OR
VISIT WWW.SYS-CON.COM

SYS-CON MEDIA
The World's Leading i-Technology Publisher

clients, on the other hand, have to do some heavy lifting in the sense that they have to process XML messages.

Being ubiquitous – or the adherence to the anytime, anywhere, and anyhow paradigm – is the real driver for enterprises to embrace mobile computing. Mobile clients are highly suited to consuming services and content provided by enterprise portals and other intranet applications. Tailoring content based on a device's capabilities is known as transcoding. Two approaches can be followed for handling the presentation for mobile devices, based on device capabilities and the network service available.

Design-time, or static, transcoding has different versions of the same content developed and placed in the file system of the Web server. The main problem is in selecting the right version of the content for a particular user with a particular device. The disadvantage of this approach is that a number of disparate pages and applications have to be developed and maintained; with an increase in the number of devices to be supported, this becomes a daunting task.

Runtime, or dynamic, transcoding enables the clear separation of the creation of content from the creation of different presentations (see Figure 2). Here, content is massaged based on the presentation requirements for a particular user on a particular device and on a particular type of network service. Dynamic transcoding uses style sheets to convert XML documents into desired presentation relevant to a particular device or translation from HTML into various markup languages.

These approaches can be combined to meet the requirements of particular applications. Portals are the centralized entry point into a system from which a user initiates a Web browsing experience, and can be seen as a delivery mechanism for content transcoded for specific devices.

Controlling Access

Portal technology involves both authentication and authorization as security mechanisms for ensuring the credibility of users and the user's privileges. Single Sign-On (SSO) is one way of enabling this. Once the user's credentials are validated and the application access profile is checked, no further authorization restrictions are placed on the user. This system ensures dependability and ease of maintenance, while providing a high level of convenience for a user who wants to access multiple applications.

Portals can use Web services to integrate disparate applications and systems, each replete with specific authentication mechanisms. An obvious application of SSO to Web services for portals could lie in the management of authentication credentials into one scheme.

Security Access Markup Language (SAML) enables the encoding of authentication and authorization information in XML format. Under this standard, a Web service interface can request and receive SAML Assertions from a SAML-compliant authority to authenticate and authorize a service requestor. SAML can then pass credentials off to multiple systems and to SSO solutions for controlling access to Web services and portal content.

The Open Solution: OASIS WSRP and OASIS Provisioning

The OASIS WSRP Technical Committee (www.oasis-open.org/committees/wsrp) addresses the use of Web services as pluggable components for portals. These services will enable businesses to provide content or applications to portals without massaging the same for specific form and presentation requirements, thus providing a cost-effective and efficient way to integrate and deploy Web and portal systems.

A portal can find and bind remote portlet Web services using a portlet proxy just as it would through any portlet. This proxy works on the standard Web services request-and-response mechanism where a SOAP request can be built, forwarded to the WSRP service, and the response then delivered to the portal. The WSRP service can then be easily published for discovery and consumption for administrative portal users through dynamic integration. Through this initiative, remote portlet Web services can be implemented for both Java and .NET technologies. WSRP will use standards such as SOAP, UDDI, and WSDL.

The OASIS Provisioning Services Technical Committee (PSTC) (www.oasis-open .org/committees/provision) defines an XML-based framework for exchanging information between provisioning service points – Service Provisioning Markup Language (SPML). The technical committee is developing an open specification detailing the semantics for provisioning service points to exchange requests relating to the managed provisioning service targets. SPML requests will facilitate the creation, modification, activation, suspension, enablement, and deletion of data on managed provisioning service targets. This specification is expected to facilitate SPML exchanges such as provisioning requests between provisioning service points within one or more organizations provisioning requests between provisioning service points hosted by third-party providers, aggregators, and ASPs; and provisioning requests between provisioning service points with support for chained or forwarded requests.

The J2EE Solution: Java Community Process JSR 124

Client provisioning is an end-to-end paradigm by which client applications, Web services, and content that is stocked or aggregated can be administered – or vended – to a variety of client devices. There are many such devices available, varying in memory footprint, size, functionality, and features. Whether these devices are built on J2ME, J2SE, or even non-Java platforms, a number of challenges are presented in their administrative tasks. Management, upgrade, and usage tracking of such devices are replete with technical challenges which the client provisioning paradigm seems to address by providing an infrastructure to mediate and manage the client applications, Web services, and content delivered.

Sun Microsystems' Java Community Process (JCP) features the JSR 124 J2EE Client Provisioning Specification (http://jcp.org/en/jsr/detail?id=124). Based on this, client provisioning can be defined as activities of advertising client services to a client device, and to the process of delivering the service to the device.

The main components of a client provisioning infrastructure are comprised of a Provisioning Portal and APIs that enable delivery of client applications; Web services and content based on various policies using J2EE components; a Provisioning ARchive (PAR), which is a standardized file format for handling the packaging of applications, Web services, and content and their delivery to the portal; and a Service Provider Interface (SPI) that is extensible and enables the provisioning of new types of Java and non-Java devices by the provisioning server. The SPI consists of a provisioning adapter – software that hides the system details for provisioning a particular type of device and enables the secure discovery and delivery of client applications.

Applying the Client Provisioning Paradigm

Consider a traditional vending machine from which users can select from a display of available items and have their selections delivered. A vending machine is an infrastructure that is accessible to multiple device types, provides services to multiple device types, delivers content tailored to a device, must be extendable across enterprises, eases development for extending services, and enables a scalable and open architecture. The services of such an infrastructure may include presentation, personalization, content management, enterprise application integration, and page transformation.

Derived from the client provisioning paradigm, the vending machine consists of a service developer interface, a service consumer interface, and a service vendor interface (see Figure 3). Through authentication, enrollment, metering, billing, and managing operations that control the behavior during use, applications, Web services and content may be vended.

A synopsis of the end-to-end client request, discovery, stocking, delivery, and administration, comprehensively termed as vending of a Web service in the context of the vending machine paradigm, may be broken into three steps: development and deployment, enrollment and subscription and usage (see Figure 4).

Conclusion

In our previous article ("Extending Web Services to the Real World – Using SOAP to Access a Mobile Device," ***WSJ***, Vol. 3, issue 3), you saw how a Web service is accessed from a SOAP client running on a J2ME mobile device. In this article we looked at accessing a Web service through a centralized portal, using the Client Provisioning paradigm to manage access based on a user and device profile, and usage based on billing and metering. We also looked at how specific client presentation requirements can be transparently dealt with using transcoding techniques, thus enabling the use of any device to access back-end functionality. By applying the client provisioning paradigm and the Java Vending Machine analogy to a device-independent portal based software solution, we can demonstrate ubiquity, or the discovery and delivery of applications, Web services, and content – anytime, anywhere and anyhow.

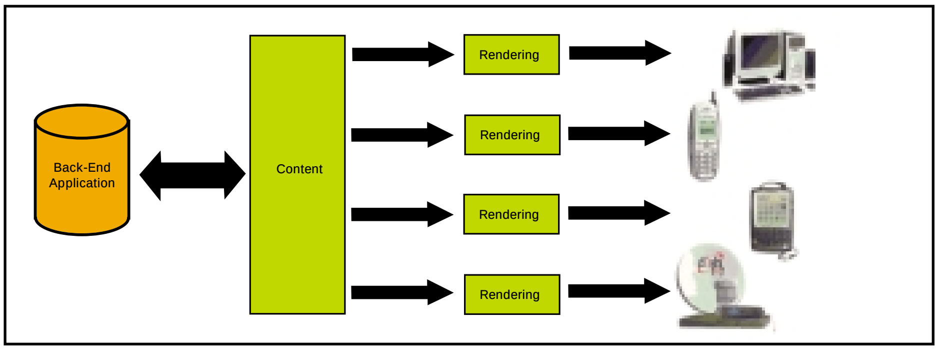


FIGURE 2 Dynamic transcoding

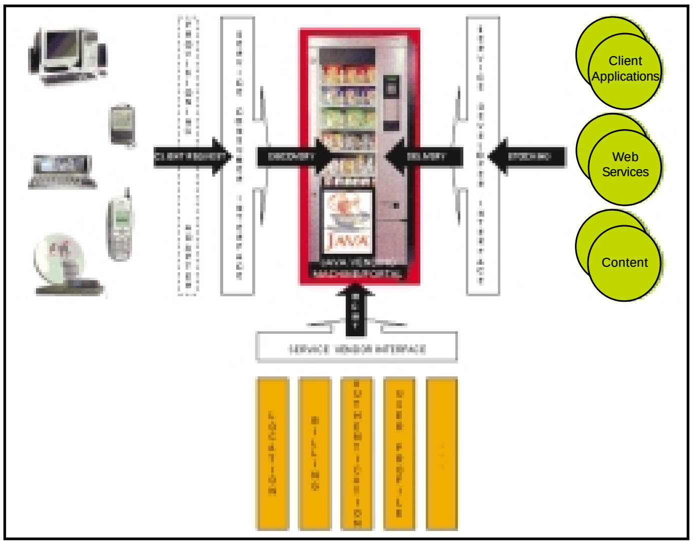


FIGURE 3 The Java Vending Machine

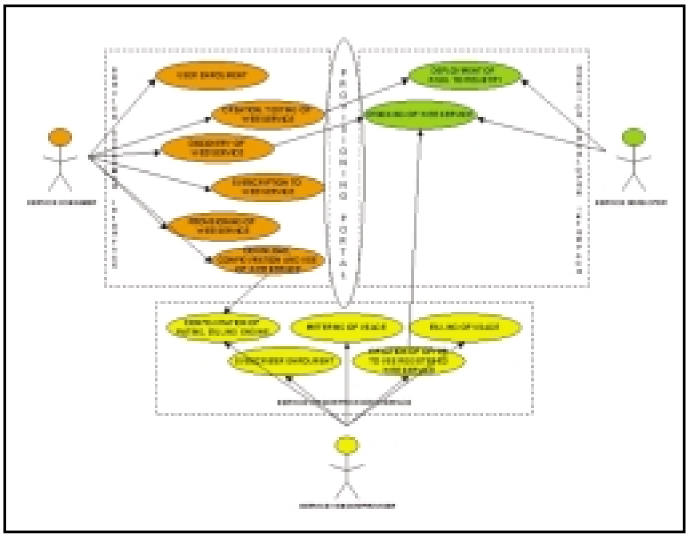


FIGURE 4 Use-case scenario showing interaction between the interfaces

Altova Offers Free XML Document Editor

(Beverly, MA) – Altova Inc., producer of XMLSPY, has announced that their XML document editor product, AUTHENTIC 5, will now be offered to the public through a free software license.

AUTHENTIC 5 allows business users to create and edit content through a Web-enabled interface that resembles a word processor. AUTHENTIC 5 features full support for standard Internet protocols and file transfer interfaces, including WebDAV and HTTP; a browser plug-in that enables a business user to access and edit XML content on the Web; spell-checking capabilities in 14 languages; real-time document validation; and built-in templates for over 15 industry-standard XML content formats. AUTHENTIC 5 can be used in conjunction with the leading XML content repositories. In addition, AUTHENTIC 5 can easily be deployed in custom XML editing applications as a custom control for ASP.NET.

www.altova.com

SonicMQ 5.0 Sets the Standard for Enterprise Messaging

(Bedford, MA) – Sonic Software has announced the availability of SonicMQ 5.0. With this release, Sonic sets the standard for enterprise-class messaging, and extends its competitive advantage with new clustering, management, and security features designed to meet the requirements of the most demanding global IT organizations.

SonicMQ 5.0 features enhancements in high availability and scalability, configuration and operational management, and security.

www.sonicsoftware.com

Microsoft and AmberPoint Announce Strategic Agreement

(San Francisco) – Microsoft Corp. and AmberPoint, Inc., announced they have reached an agreement to build and deliver a combination of tools and management technology intended to help customers manage distributed applications based on the Microsoft .NET Framework. The combination of AmberPoint's strengths in Web services management and Microsoft's leading platform, tools, and management solutions will enable IT organizations to more effectively build and manage Web service–based solutions.

The .NET versions of AmberPoint Management Foundation fully utilize the XML Web services system libraries in the Microsoft .NET Framework and streamline Web services management through point-and-click operations.

www.amberpoint.com, www.microsoft.com

Accenture Platform Helps Businesses Manage Large-Scale Web Services Projects

(New York) – Accenture has introduced a software platform designed to make it easier for the company and its clients to manage, build, and roll out large-scale deployments of Web services.

An integrated "toolbox" of reusable software components and services that span the entire software development cycle, the Accenture Web Services Platform is designed for enterprises that need to manage and deploy projects across multiple geographies and subsidiaries or partner companies. The platform can help designers and project managers define business problems and processes and test and catalog results. It also provides programmers and developers with a single work environment that integrates best-in-class tools and ready access to pretested, preassembled building blocks.

www.accenture.com

The Mind Electric Raises Bar with Major New Release of Web Services Platform

(Dallas, TX) – The Mind Electric, a provider of software for service-oriented architectures, has announced the immediate availability of version 4.0 of its GLUE Professional Web services platform. GLUE is a fast, comprehensive, easy-to-use Java platform for creating and deploying applications with Web services, JSPs, and servlets.

GLUE Professional 4.0 provides businesses with more options, flexibility, and control in implementing a service-based architecture to gain control over and extend the value of business applications while improving the availability, reliability, and trustworthiness of Web services offerings to their users. The system continues to deliver on The Mind Electric's reputation for providing a robust feature set that includes high performance; deep integration with J2EE; broad standards support for XML, SOAP, WSDL, and UDDI; and strong interoperability with .NET.

www.themindelectric.com

HP Unveils Software, Services, Standards Framework for Web Services

(Orlando, FL) – Chairman and Chief Executive Officer Carly Fiorina outlined several HP investments and technologies to help customers address key management issues that may inhibit the adoption of Web services. During her keynote address at the eighth annual BEA eWorld user conference, Fiorina focused on the critical role Web services management will play in overcoming IT complexity, unifying systems, and accelerating a return on IT investments.

HP also announced the creation of a dedicated Web Services Management team to extend the HP OpenView software suite for simplified Web services management. The team, reporting to Nora Denzel, senior vice president of software, will lead HP's efforts to help customers provision and manage Web services with the choice and flexibility to adopt future innovations, including a consistent management interface for J2EE and .NET components. The team also will continue HP's active work in Web services standards.

www.hp.com

ObjAcct Business Systems Announces Web Services Accounting Solution

(Bowling Green, OH) – ObjAcct Business Systems, LLC has announced the release of its ObjAcct XML Accounting Server. Based on Microsoft .NET, the system is an n-tier SOAP and XML Web services accounting solution designed to be integrated into portals and ASPs, embedded in vertical market systems, or implemented as a custom corporate solution. The system includes ObjAcct Web Ledger, a complete Web-based accounting application, and ObjAcct XML Web Services, a complete XML-based API to the ObjAcct Transaction Server.

The ObjAcct XML Accounting Server is licensed both as a "boxed" server product that can be hosted and implemented by licensees within their own data center or network, and as a subscription ASP product hosted by ObjAcct Business Systems. The ASP License allows licensees to integrate the system into their product offering and offer the solution to their customers. The product includes ObjAcct Site Manager, an application suite for managing and maintaining the system in an ASP environment.

www.objacct.com

WSJ ADVERTISER INDEX

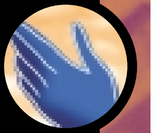
ADVERTISER	URL	PHONE	PAGE
Altova	http://xmlj.altova.com/authentic5	978 816-1600	11
Global Knowledge	www.globalknowledge.com	1-800-COURSES	15
IBM	www.ibm.com/websphere/portalplay		CII
Java Developer's Journal	www.sys-con.com	888-303-5282	57
JDJ Store	www.jdjstore.com	888.303.JAVA(5282)	32
JDJ Workshop	www.sys-con.com/education	201 802-3058	35
Linux Business & Technology	www.sys-con.com	888-303-5282	53
MindReef	www.mindreef.com/wsj	603.465.2204	4, 5
Parasoft	www.parasoft.com/wsj2	888-305-0041	7
PriceWaterHouseCoopers	www.pwcglobal.com/tech-forecast_syscon		29
Sitraka Jclass Server Chart	http://java.quest.com/jcsc/ws	800-663-4723	9
Sitraka Jclass Server Report	http://java.quest.com/jcsr/ws	800-663-4723	CIII
Spiritsoft	www.spiritsoft.com/climber	508 473-3227	17
Swingtide	www.swingtide.com/testdrive		CIV
SYS-CON Media	www.sys-con.com	888-303-5282	44,45,49
Web Services Journal	www.sys-con.com	888-303-5282	39

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. This index is provided as an additional service to our readers. The publisher does not assume any liability for errors or omissions.

IN THE NEXT ISSUE OF WSJ...

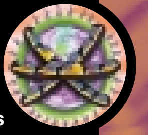
Focus: EAI/ERP

Putting Data and Business Process Integration in Context



While seamless integration and across-the-board automation may be highly visible IT goals, the business process needs of employees, customers, business partners, and suppliers are equally important. This article will examine enterprise application integration in the context of technology and business process relationships.

Business Processes: Turning integration upside down?



The major ERP and other enterprise application vendors are feverishly working to deliver predefined business processes, which can be deployed regardless of what technology components are used to execute the business logic. This is a significant departure from enterprise integration approaches of the past, essentially addressing the problem top-down from the business problem, as opposed to bottom-up from the technology.

Plus:

Introducing WS-Coordination

BPEL4WS, WS-Transaction, and WS-Coordination form the bedrock for reliably choreographing Web services-based applications, providing business process management, transactional integrity, and generic coordination facilities respectively. This article shows you how a generic coordination framework can be used to provide the foundation for higher-level business processes.

Strategies For Achieving Real-time Application Integration With Web Services

Package application vendors have been agressively pursuing Web services to reduce the cost and complexities of integrating their applications in the context of a business process inside and outside the enterprise. We'll look at the approaches being taken and a call to action for pursuing a business process approach to leveraging the power of Web services to integrate cross-business applications using Web services.

WebServices JOURNAL .NET J2EE XML

Message-Centric Web Services vs RPC-Style Web Services

Dave Chappell

Dave Chappell is vice president and chief technology evangelist for Sonic Software, the leading provider of integration products and services for the real-time enterprise. Dave is coauthor of Java Web Services *(O'Reilly & Associates, 2002),* Professional ebXML Foundations *(Wrox, 2001), and* The Java Message Service *(O'Reilly & Associates, 2000), and a frequent contributor to* Web Services Journal.

Message-centric vs RPC-style Web services is a long-standing debate and bone of contention regarding the proper use of Web services technologies. Early renditions of SOAP and XML-RPC were all about providing RPC-style interactions…in fact, that's all that was supported, so there really wasn't much choice in the matter.

RPC-style interfaces have their advantages: immediate gratification of request/response, and a programming model whereby remote procedures are exposed in a way that mimics the underlying object architecture of the applications concerned, allowing a developer to make "normal"-looking method calls in the native language. Are these benefits really worth it?

Developers, system architects, standards bodies, and vendors alike have all come around to the idea that message-style interactions between Web services are an attractive alternative to RPC-style Web services. It's a real pleasure to see that Web services specifications and specific implementations have matured enough to make message-centric Web services a reality. Let's examine some of the drivers that have brought on this trend:

In a business environment where Web services are being deployed as a means for linking together businesses in a value chain, message-style interactions model the way that businesses really interact with one another. These interactions have three main characteristics: loosely coupled interfaces, loosely coupled interactions, and long duration conversations. Let's use a purchase order as an example, and look at it for what it is – a document presented by a buyer to a supplier that signifies the buyer's intent to acquire, and eventually pay for, an item or list of items from the supplier.

Loosely coupled interfaces: A PO document contains all of the information necessary to fulfill the order and allow the payment process to begin – the billing information, shipping address, line item information, quantity, price, etc. A buyer needs only to send that information to a supplier once as one XML document, encapsulated inside of a message. The processing of the PO involves many discrete operations – credit check, contacting other suppliers, and so on. Loosely coupled interfaces means that each application or service in the process acts upon a self-describing XML message, extracting or modifying just the parts of the message it needs.

In a tightly coupled RPC environment, each application has to know the intimate details of how every other application wants to be communicated with – the number of methods it exposes, and the details of the parameters that each method accepts. Multiply the number of applications and services that make up an extended enterprise by the number of interfaces that each one might want to expose, and then square that. That's roughly the number of interfaces that you will have to create and maintain over time.

Loosely coupled interactions and availability: An application sends a message to another application via a message-style Web service invocation. This is part of a larger multistep process. Using message-style interactions, each step in the process is asynchronous and autonomous. If the *n*th receiver in the chain is not available, the initial sender is not concerned. If reliable messaging is being used in the form of JMS or WS-Reliability, then message persistence, retries, and redelivery can happen at the transport level without the applications or services even being aware that this is taking place.

In contrast, a tightly coupled RPC environment requires that all parties be available in order for the entire operation to be successful. Now take that scenario and magnify it across hundreds or thousands of interconnected endpoints and you'll see how quickly it can become an unwieldy proposition.

Long-duration conversations: The purchase order very likely was a follow-on message from a request for a price quote. The fulfillment of that order may take days, weeks, or months before all the items in the order are shipped and billing can take place. Having complete, self-describing documents as the basis for state persistence makes the separation of time irrelevant.

In contrast, a fine-grained RPC- style Web service invocation will probably not have all of the context required to span time, or even a single application session.

Data transformation and routing: Every application has its own way of internally representing data, and not all business partners are going to agree on the same dialect of XML. The availability of a self-contained XML document at all times lends itself very well to utilizing XSLT for translation to target data formats, and XPATH for routing of messages based on their content

When all is said and done, the benefits of message-centric Web services far outweigh the perceived advantages of RPC. So get on board, and get messaging!

Sitraka

www.java.quest.com/jcsr/ws

SWINGTIDE

www.swingtide.com/testdrive