FREE EXPO PASS! $75 VALUE Page 61

WEB SERVICES EDGE WEST OCT. 1-3

September 2002 Volume 2 Issue 9

WebServices JOURNAL

.NET J2EE XML

WSJ2.COM

FULL CONFERENCE PROGRAM

THE LARGEST WEB SERVICES CONFERENCE & EXPO IN THE WORLD

WEB SERVICES EDGE

INTERNATIONAL WEB SERVICES CONFERENCE & EXPO

McENERY CONVENTION CENTER, SAN JOSE, CA

INTEGRATING APPLICATIONS ▶▶▶ CONNECTING ENTERPRISES

CONFERENCE: October 1-3, 2002

EXPO: October 2-3, 2002

CALL TODAY!

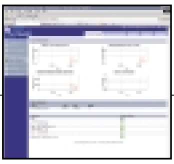
SAVE $200

▶▶▶ INSIDE PAGE 59

PAGE 26

Web Services and WIRELESS MESSAGING

Innovations in field workforce management



RETAILERS PLEASE DISPLAY UNTIL OCTOBER 31, 2002

$6.99US $7.99CAN

0 71486 03420 9 09

SYS-CON MEDIA

IBM

www.ibm.com/websphere/integrate

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

One Year Later

Written by
Sean Rhody

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

We aren't a news magazine, and we don't cover current events except as they relate to our industry, so I had to think long and hard about what to say in this edition. Nevertheless, I decided that at least in my editorial, we would remember September 11th.

Our premier issue was launched the week before the attack. As you may know, the publishing company is in New Jersey, a few miles from the New York border. And I live in New Jersey and work there, as well as in New York city itself.

On the 11th, I was driving to West Orange, which is about 20 miles away from NYC. There's a high ridge on the way to my office, from which you can see the skyline, or at least a bit of it, on a clear day. That day, I saw a plume of smoke at a little after 9 a.m., as I crested the ridge. My cell phone sprang to life, with friends who were watching the news telling me of a plane crash, and warning me not to go into the tunnels to the city, as I often do. Later that day, I took one of my colleagues to my house, since it was impossible to return to NYC that day.

We recently held our second annual Web Services Edge conference at the Javits Center, which attracted over 7,000 attendees. As the tech chairman, I was both honored and humbled to be in the city again. Our first show was September 24-27, less than two weeks after the attack. Although we had some cancellations, we carried on, as Rudy Guiliani asked, with "business as usual." And 2500 people turned out to carry on with us.

During the week, I went down to Ground Zero for the first time. It was strange, and I couldn't quite grasp the enormity of it all. Then I got my bearings, and saw that a park I used to walk through on my way to Wall Street had been turned into a parking lot for emergency vehicles. The whole world tilted for a moment, and I stopped walking and just stood there. The friends who were with me even got concerned. But it was only then that the full impact hit me. I knew where I was, and knew that the street I walked on, close to the gigantic hole in the earth, was the same street that I had emerged to countless times from the PATH train.

I often work in an office above Grand Central Station. And I see the signs, the postings from people who've lost loved ones, in the corridors of the station. It's a moving, chilling experience. I see the notes, taped to pictures of lost relatives, from others who knew them, and who wanted to let them know that they too grieved for the loss.

We at the magazine also grieve. Each of us knew someone who was there who survived, or who didn't. My sympathies and condolences go out to every family who lost a loved one during that merciless day, be it in New York, Washington, or the woods of Pennsylvania.

I can only say that we here at SYS-CON, like many others in the United States, have learned to treasure our freedom even more dearly. I value highly, and closely guard, the right to express my opinion, and for others to do the same. I'm proud to live in the United States of America, where we are free to speak our minds, and willing to give our lives for such freedom. Thank you to those who have given all, our servicemen and women, and all who have striven to make our country safe for freedom.

God Bless America.

Actional

www.actional.com/whitepapers/wsjblues

Written by Andrew Bibby and Cesar Brea

Business Process Reengineering Through Web Services

A user-experience perspective

AUTHOR BIOS:

Andrew Bibby is VP, Client Services and Alliances, Razorfish. Andrew ensures that value is driven by major alliances throughout the client-services organization. He has 16 years of experience developing and implementing both project and relationship management methodologies in the professional services industry.
ANDREW.BIBBY@RAZORFISH.COM

Cesar Brea is senior vice president of Sales and Marketing at Razorfish. He leads Razorfish's marketing and sales initiatives, including strategies and programs to position and communicate the Razorfish brand and service offerings to Fortune 1000 decision-makers and market influencers. Prior to his current position, Cesar worked with Razorfish as a consultant, adding value to its sales and marketing efforts.
CESAR.BREA@RAZORFISH.COM

Promise:Web services will be revolutionary!
Reality: What's a Web service for? What's the investment case?
Problem: Most discussions of new solutions based on Web-services architectures focus on them from an IT-centric perspective
Solution: An alternative view that helps to envision new solutions and prioritize investment in them starts with understanding business processes end-to-end from the perspective of user experiences. Specifically:
- Are users' choices usefully expanded?
- Is the aggregation of information automated sufficiently to save valuable time?
- Do integration savings get channeled into features, content enhancement, and design improvements that raise user adoption and retention?

A Better Noise Filter

Web services, XML, UDDI, SOAP, WSDL, .NET. Today it seems that anyone even thinking of investing in technology to solve a business problem is immediately overwhelmed by a cacophony of alphabet soup–like acronyms with the common promise of... well, just what isn't entirely clear. But "this is going to be really big, so you have no choice but to pay attention."

Many clients have asked us for help in making sense of these new technologies. They ask, "What possibilities – both opportunities and threats – do these technologies imply for us?", and "What should we do about them?".

We think about questions such as these "outside-in." To what business process will a new technology be applied? How

Sonic Software

www.sonicsoftware.com/websj

will it change the experiences of the various users who are involved in the business process? What economic impact could those changes have?

Web Services for Business People

Just what the world needs: another Web services primer! But unfortunately, to imagine what a new technology makes possible in any original way requires at least a basic understanding of how it works.

Consider this analogy:

The Human Experience: You are a tax accountant who specializes in the Andorran tax code. You need a job. Your friend helps you write a summary of your experience in English using a conventional resumé format and posts it on Monster.com. With luck, an employer browses the site, you get a phone call or email, you're hired, and you have a long and lucrative career.

The Machine Experience: You are an Andorran tax accounting program. Your authors write what you do in a programming language, following a protocol or format called SOAP, using a conventional structure called "WSDL (Web Services Description Language)." They post you on a "UDDI (Universal Description, Discovery, and Integration) registry." With luck, a financial accounting package that doesn't calculate Andorran taxes itself browses this registry, identifies you as a relevant "Web service," and you get a "Remote Procedure Call" or "RPC" (itself expressed in SOAP). You return a valuable answer, and earn your subscription fee or "micropayment."

Continuing this analogy, if your authors are Microsoft customers they will use tools such as Visual Studio .NET to write in Visual Basic just as Microsoft Word is used to write English. SOAP is a protocol – an agreed-upon format for transmitting data between a Web service and the application that would like to use it (in this case, the financial accounting package). Microsoft BizTalk servers are the traffic cops that log incoming RPCs/requests and route them to appropriate Web services.

The bottom line: With the proliferation of valuable content and applications made possible by the Web, there is a huge opportunity associated with automating the discovery and exchange of those assets. Just as people use tools and conventions such as search engines and "About Us" pages, machines need conventions to find each other and communicate effectively. A Web service is thus a computer program exposed to and used by other computer programs across the Web using pre-agreed conventions as its lingua franca.

Great! Now What?

Okay, we now understand there is a commonly agreed way to get programs talking to other programs over the Internet. Which programs and why?

People and programs work together toward various objectives through "business processes." For example:

- A customer interacts with salespeople and/or a Web site to identify and purchase a product to meet a particular need.
- A manufacturer interacts with various suppliers to line up various resources.
- A channel partner checks on inventory to make sure it can fulfill orders it has taken on behalf of the manufacturer.

Each of these processes typically encompasses several major activities. For example, selling something involves:

- For a successful sale, a customer first has to perceive a need, get educated about product options, order and receive the chosen product, and use it successfully so it won't be returned.
- To acquire the inputs it needs to produce a product, a manufacturer first must identify, screen, engage, and transact with its suppliers.
- Similarly, a channel partner must qualify itself into the manufacturer's distribution program (or vice versa), and then get "hooked up" appropriately before it can view inventories. The partner may even need to attain certain performance levels before it will receive allocations of popular but scarce products.

Each of these activities is both enabled and constrained in certain ways by the technologies applied to it. Consider these three examples:

- Our customer may have received a CD from a financial planning software company with a "configurator" application to help him think through his needs and determine which of the company's offerings might best fit those needs. Let's assume that unfortunately the customer left the CD at home and he's on vacation. There happens to be a new Web-based version of the program that he could use available at a nearby Internet café, but the program requires up-to-date information from various financial services providers, and is not yet able to reach across the Web to get the information directly from them.
- Our manufacturer had joined a consortium to sponsor and exchange information and resources, and spent many millions implementing the software and recruiting suppliers. Unfortunately, adverse selection lowered the quality of the supplier pool, and the exchange never aggregated enough demand to attract the best suppliers in the market. Now our manufacturer needs to create closer ties with his preferred vendors, but must find an inexpensive and nonproprietary way to connect with them.
- Our value-added reseller combines products from multiple plants in our manufacturer's network to create solutions for her customers. Tracking inventories online allows her to do some "critical path planning" so she doesn't take assets onto her loading docks and balance sheets in advance of the full solution being ready to roll. Unfortunately, the manufacturer's plants, brought together through many acquisitions over the years, have implemented EDI links to their distributors inconsistently and at great cost to all parties. The result is that the process of managing a "bill of materials" for the final shipment is manually intensive and error-prone.

A "Web services" approach to the supporting technologies for each of these examples could change the user experience both within each activity and across the entire process. For example:

- If the financial services vendors' applications were to be made available as Web services, the Web application that our customer visits from the mythical Internet café could determine the impact that day-to-day changes in his investment portfolio would have on the length and amenities of the yacht he can afford to buy.

- By exposing elements of their respective supply-chain planning platforms as Web services based on neutral protocols, our manufacturer and its suppliers can exchange demand forecasts and inventory information automatically, without committing to expensive and proprietary formats (assuming their industry already has an agreed-upon XML "DTD" or dictionary that defines which tags stand for what data elements).
- Just as our manufacturer could work with its suppliers, our reseller can work with plants from which it sources parts to "wrap" and "expose" inventory applications as XML-based Web services that the reseller's tracking applications can "consume." If the reseller's own supply network comes up short, it can "browse" UDDI registries for other vendors that meet its needs.

What is the value of these solutions? First-order effects include:

- ***Lower integration costs:*** A good rule of thumb from the ERP implementation world of the early '90s was that integration costs would run 3x original license costs. In the more significantly networked Internet world of the early 21st century, some top analysts estimate that this multiple has increased to 10x. Since not everyone can afford 10x, a lot less integration will take place unless an alternative, less-expensive approach can be found. Enter Web services, based on a lingua franca without proprietary technology requirements.
- ***Expanded choice:*** Automated search increases the likelihood of finding an alternative source of supply, distribution, or information.
- ***Increased service performance throughout a given business process:*** Because supply, distribution, and information options have increased, waiting time for any item in a business process "queue" will decline.

Second-order effects are equally powerful:

- Employees are freed from manual and mechanical search and aggregation tasks and can focus on service and innovation.
- Customers experience greater choice, better and faster service, and lower costs when they conduct business with vendors whose infrastructure is enabled through Web services.

From Here to Where?

How do we go about realizing this glorious future? There are three questions to be answered:

1. What business process offers the greatest return on investment from improvements made to it?
2. What specific change in a given business process drives the most improvement in performance?
3. Is a fix based on Web services the most appropriate solution?

Answering the first question requires some strategic research, such as:

- Modeling the value of improvements in performance to customers (or other consumers of the outputs of a given business process)
- Benchmarking how such improvements would position your organization relative to competitors

Good answers for the second question can be found through basic business process analysis and redesign:

- Mapping steps in a process, and associated performance metrics (time, cost)
- Understanding the resources (material, human, technical) employed at each step

As for the third question – "Are Web services the answer?" – it makes sense to look beyond what you do see to what you don't:

- What valuable information that might help answer customer questions or issues is not provided due to cost or availability limitations?
- What segments of suppliers or distributors are not included in current networks due to cost or technology?

Practical Magic

Web services are subject to network effects. The more of them that exist, the more valuable they are, and the challenge is that we are just at the dawn of the Web services era. Since the use of Web services is a fairly new concept, it is also still relatively expensive to implement. And since the pool of available Web services and users is still nascent, the return on the initial investment is limited.

Fortunately, major information technology infrastructure vendors understand these issues and are lowering the cost and complexity barriers to getting started. Major telecom vendors are getting into the UDDI registry business – essentially Yellow Pages for computers. Early pricing is quite favorable. Vendors such as Microsoft and Sun Microsystems have developed frameworks for building and deploying applications so that the last step – exposing them as Web services that can be consumed across a network – occurs automatically. Microsoft's .NET family of products is probably the best example of this.

What tools you use depends a lot on where you start. If you're already a Microsoft shop, you'll likely want to stay that way in the .NET world. If you're not, you will need to weigh the advantages of Microsoft's approach versus other vendors such as Sun or IBM, taking into account the cost of required infrastructure changes. The good news is that the steep performance improvement "curves" in the IT business drive significant cost-improvement opportunities in infrastructure over relatively short time horizons. Therefore, such infrastructure transformations are not only realistic but also often attractive.

Parting Thoughts

You've run focus groups; customers tell you they love the improvements you're proposing. You've benchmarked: none of your competitors would, or could, get close. You've mapped processes and found the "Herbie," or the bottleneck.

You've architected a cutting-edge, Web services–based solution.

On paper it sounds brilliant. But as we have all discovered, in practice there is a lot more to "getting the horse to drink." Chief among these factors is usability, both in the narrow sense of a tool itself, and viewed against a backdrop that reveals how well the tool would fit in a person's everyday life.

In short, don't forget that while improving the user experience may start with major performance gains through Web services–enabled business process redesign, it's unlikely to end there. A smart investment program will consider ancillary usability issues to maximize available returns.

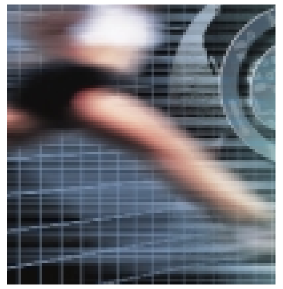
Sitraka

www.sitraka.com/jclass/ws

Written by Helen Thomas

Identifying and Eliminating Web Services Transaction Bottlenecks

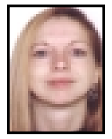
Use caching to decrease delay



Enterprises are beginning to develop and deploy applications with Web services, a standards-based approach to achieving application interoperability. Web services technology is enabled by a set of standards that specify description, discovery, and invocation via the Internet.

Web services has the potential to remove the long-standing barriers between software applications, simplifying partner integration, cutting development time and costs, and extending the lifetimes of legacy systems.Unfortunately, the Web services model isn't without drawbacks; first and foremost are performance and scalability. Through close analysis, it's apparent that the technologies that underpin Web services involve significant overhead. If these inefficiencies aren't addressed, the viability of the Web services paradigm could be threatened.

This article examines the bottlenecks inherent in existing Web services technologies and explores optimization techniques that can be used to mitigate these bottlenecks.



AUTHOR BIO:

Helen Thomas is the CTO and cofounder of Chutney Technologies, a software company that develops solutions to improve the scalability and performance of enterprise applications. She is also an Assistant Professor of Management Information Systems at Carnegie Mellon University. Helen received her Ph.D. from Georgia Tech, an M.S.E. degree in Operations Research and Industrial Engineering from the University of Texas, and a B.S. degree in Decision and Information Sciences from the University of Maryland.

HELEN.THOMAS@CHUTNEYTECH.COM

Overview of Web Services

Before delving into the bottlenecks associated with the Web services framework, it's useful to examine the details of the underlying technologies. This article assumes that you are familiar with existing Web services–enabling technologies: Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI).

SOAP is an extensible XML messaging protocol. It defines an XML encoding for function calls and their parameters, allowing applications to communicate with one another regardless of their environments. WSDL is an XML encoding for the description of Web services, specifying properties of services, such as what a service can do; how to invoke it; and where it resides. UDDI is a repository-based registry service for finding and publishing Web services.

Let's look at an example of Web services message exchange to see how these technologies work. Consider a portal application that serves stock quotes, weather, and news, each retrieved from its respective service provider. We'll use the stock quote example to examine the Web services interaction in greater detail. We'll assume that the portal has already identified the provider and has obtained the service description, i.e., WSDL. (These steps can be handled easily using the UDDI find and publish APIs.) Also, while SOAP is independent of transport protocol, we'll consider SOAP over HTTP in this article, since this is the most commonly used transport.

In the example shown in Listing 1, the portal application is the Web service consumer, and the stock quote service is the provider. The consumer and provider applications must each perform several tasks to complete the stock quote Web services transaction. The consumer executes a program that prepares a SOAP request, sends it to the provider over HTTP, receives the SOAP response, extracts the response, and performs postprocessing to convert the response into a format appropriate for consumption. Listing 2 provides sample Java code for a consumer application that performs these tasks. This example is based on the Apache SOAP implementation.

To retrieve the quote for "IBM," invoke this program using the following command:

```
>java getQuote
http://localhost:8080/axis/servlet/oper-
outer
```

SpiritSoft

www.spiritsoft.com/climber

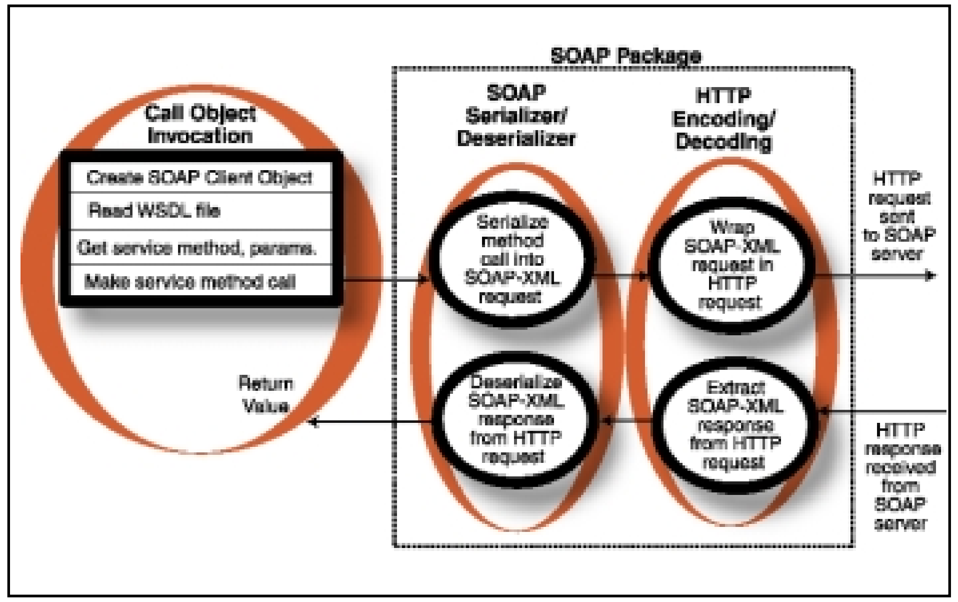

SOAP Package
Call Object Invocation
Create SOAP Client Object
Read WSDL file
Get service method, params.
Make service method call
Return Value
SOAP Serializer/ Deserializer
Serialize method call into SOAP-XML request
Deserialize SOAP-XML response from HTTP request
HTTP Encoding/ Decoding
Wrap SOAP-XML request in HTTP request
Extract SOAP-XML response from HTTP request
HTTP request sent to SOAP server
HTTP response received from SOAP server

FIGURE 1 Consumer-side process (Adapted from "SOAP Basics," www.soapuser.com)

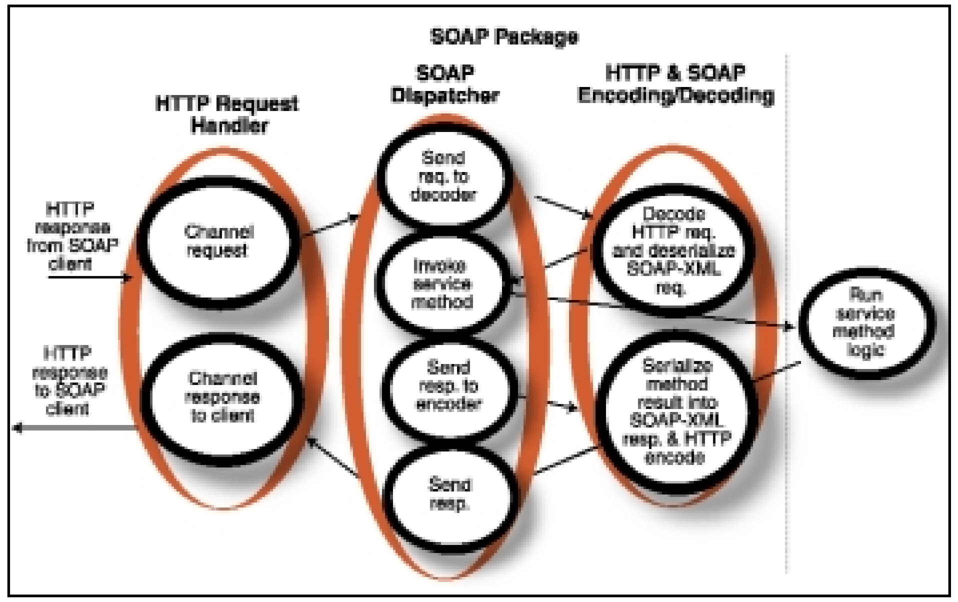

SOAP Package
HTTP Request Handler
SOAP Dispatcher
HTTP & SOAP Encoding/Decoding
HTTP response from SOAP client
HTTP response to SOAP client
Channel request
Channel response to client
Send req. to decoder
Invoke service method
Send resp. to encoder
Send resp.
Decode HTTP req. and deserialize SOAP-XML req.
Serialize method result into SOAP-XML resp. & HTTP encode
Run service method logic

FIGURE 2 Provider-side process (Adapted from "SOAP Basics," www.soapuser.com)

The input arguments represent the method name, service URL, and parameters (symbol in this example), respectively. These values are obtained from the WSDL for the service. The program first processes these input arguments, which are used to build the call object. Next, the call.invoke method prepares the SOAP request, sends the request over HTTP, and receives the response.Finally, the response is checked for errors and any needed post-processing is done. Figure 1 illustrates these consumer-side tasks in detail.

The SOAP-specific steps that must occur on the consumer side typically include the following:

- A SOAP client object is created.
- The service method and parameters are obtained. These items may be read from the WSDL file, which may or may not exist locally.
- The parameters needed for the method call are serialized into a SOAP request (XML format).
- The SOAP request message is encoded into an HTTP request.
- The HTTP request is sent to the service provider.
- The HTTP response is received.
- The HTTP response is decoded into the SOAP response message.
- The SOAP response message is deserialized into the method response format.

Figure 2 shows the steps required on the provider side. The HTTP Request Handler forwards the request to the SOAP Dispatcher, which controls the SOAP processing for the request:

- The HTTP request is decoded into the SOAP request message.
- The SOAP request is deserialized, i.e., the parameters are converted to the appropriate types needed to invoke the service method.
- The service method is invoked with the appropriate parameters.
- The service method response object is serialized into a SOAP response message.
- The SOAP response message is encoded into an HTTP response.
- The HTTP response is sent back to the consumer.

Delays in Web Services Transactions

A number of steps are required to carry out a single Web services transaction. Each of these steps adds delays to the end-to-end service response time. The specific delays associated with Web services can be identified as follows:

- ***Object instantiation:*** Object instantiation occurs on both the consumer and provider sides. The consumer, for instance, creates a SOAP client object. The provider usually creates application objects needed in executing the service method logic. Creation of such objects is costly in terms of processing and I/O operations. Furthermore, object creation increases the burden of memory management, i.e., garbage collection in Java environments, which can severely impact application performance.
- ***Serialization/Deserialization:*** Serialization and deserialization occur on the consumer and provider sides. These tasks require parsing operations, adding delay.
- ***Encoding/Decoding:*** Encoding and decoding occur on the consumer and provider sides. These steps require string processing operations, adding delay.
- ***Service method invocation:*** Service method invocation occurs only on the provider side. In addition to object creation, this step often includes numerous other tasks, such as I/O calls to databases and data transformations. In addition, the multilayered application-design approach followed by most enterprises typically incurs cross-tier communication costs since the applications that create objects often involve several layers of nested callouts.
- ***Network transmission:*** The exchange of SOAP messages between the participating applications occurs over a network. Regardless of whether these applications reside locally or remotely, each Web service exchange requires network communication. Thus, for each service invocation, communication between the two applications requires traversal of a network protocol stack (e.g., TCP/IP), which adds to the service response–time latency. This problem is exacerbated by the increased bandwidth requirements of SOAP, which is a text-based protocol.

It's important to note that each step must occur for each Web service request. Referring to our portal example, this means that for each request of the portal page, these steps must occur for each of the portal page elements (stock quotes, weather, and news). As the number of concurrent requests increases, these delays can cause serious performance and scalability problems.

Web Services Optimization Techniques

There are few solutions currently available to address Web services bottlenecks. One promising technique is to employ various types of caching in order to bypass many of the SOAP-related operations. I discuss three broad types of Web-services caching: consumer-side caching, provider-side caching, and gateway caching.

Consumer-Side Caching

Caching the results of Web services requests at the consumer side can reduce the end-to-end service response time. This type of caching has been suggested for Java applications as well as for .NET applications. There are different ways to implement such caching mechanisms. For instance, the cache itself can be an in-process or an out-of-process cache. An in-process cache is relatively easy to implement and is most appropriate for applications running on a single application server. An out-of-process cache allows for greater scalability and is most appropriate for applications running on multiple application server instances. However, out-of-process caches are more complex to implement and maintain.

It's useful to look at an implementation of consumer-side SOAP caching. As an example, I look at an implementation that utilizes the Cache Management design pattern for caching operations. This example is based on an implementation that utilizes the Business Delegate design pattern. For the sake of brevity, I consider only the caching portion of this implementation in this article. Listing 2 shows the cache methods for inserting into and retrieving items from a local cache.

We use the StockQuote service example to illustrate how consumer-side caching can be incorporated. Listing 3 shows an alternative version of the getQuote consumer code. This code checks the local cache for the object before invoking the SOAP call. If the object isn't found in cache, the SOAP call is executed and the returned SOAP response object is inserted into the cache. This example uses the symbol parameter as the cache key.

By caching SOAP response objects at the consumer side, the serialization, encoding, service invocation, and decoding steps shown in Figure 1 can be bypassed.

Provider-Side Caching

On the provider side, recall that the HTTP Request Handler channels SOAP requests to the SOAP Dispatcher (see Figure 2). Caching SOAP response objects can be implemented by modifying the HTTP Request Handler program. For instance, with a Java-based service, the HTTP Request Handler is a servlet that can be modified to invoke the cache.fetchObject method before sending the request to the SOAP Dispatcher, in much the same way as was done on the consumer side. When a SOAP response object is not found in cache, the usual chain of events happens (including service method execution), and the cache.addObject method is invoked to insert the response object into cache.

By caching SOAP response objects at the provider side, the serialization, service-method execution, serialization, and encoding steps shown in Figure 2 can be bypassed.

Conventional server-side caching techniques (e.g., application server caching and database caching) can also be employed at the provider side to reduce service-method execution delays.

Gateway Caching

With the rise of Web services, a new class of solutions has emerged – *Web services gateways*. Web services gateways serve as a centralized point through which all Web services traffic flows and provide a variety of functionality including security, monitoring, logging, and management. Although these gateways provide such value-added features, they also add delay since additional processing operations must be performed for each request. At a minimum, some type of inspection must be done for each SOAP request. In many cases, this may require decoding and deserializing the SOAP request, performing the required operations (e.g., authentication and transformations), serializing and encoding a new SOAP request, and finally, forwarding the request to its destination. As the number of requests increases, the gateway can quickly become a bottleneck.

To mitigate these delays, SOAP response objects can be cached at gateways. Some Web services–gateway vendors have incorporated some form of caching into their gateway products. However, most rely on in-process caching, which suffers from scalability limitations. Given this issue, expect to see out-of-process caching incorporated into Web services–gateway products in the near future.

Conclusion

Web services offer the potential to simplify partner integration, cut development time and costs, and extend the lifetimes of legacy systems. However, there are serious performance and scalability issues with the Web services paradigm. Before the benefits can be realized, enterprises must ensure that their infrastructures are ready to handle the inherent bottlenecks associated with Web services transactions.

Optimization techniques, such as SOAP caching, are emerging to address these bottlenecks. The three caching techniques discussed here can help reduce SOAP-related delays. The good news is that these techniques are quite complementary, since they each address different types of Web services bottlenecks. With some cooperation among consumers, providers, and intermediaries, further optimizations are possible. For instance, if the parties at the various endpoints are aware of optimization techniques employed by one another, it may be possible to eliminate certain redundant or unnecessary operations that could not be eliminated otherwise.

References

- Azim, O., and Hamid, A.K. (2002). "Cache SOAP Service on the Client Side." www.javaworld.com/javaworld/jw-03-20 02/jw-0308-soap_p.html, JavaWorld.
- Bequet, Henry. (2002). *Professional Java SOAP*. Wrox Press.
- Grand, M. (1998). *Patterns in Java*. John Wiley and Sons.
- "*XML Web Service Caching Strategies*." http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnservice/html/service04172002.asp
- Sun Microsystems, Inc. *J2EE Patterns Catalog*. http://developer.java.sun.com/developer/restricted/patterns/BusinessDelegate.html

Listing 1

```
public class getQuote {
  public static void main (String[] args) throws Exception {

    // Process the arguments.
    int offset = 3 - args.length;
    String encodingStyleURI = args.length == 3
                              ? args[0].substring(1)
                              :
Constants.NS_URI_SOAP_ENC;
    URL url = new URL (args[1 - offset]);
    String symbol = args[2 - offset];

    // Build the call.
    Call call = new Call ();
    call.setTargetObjectURI ("urn:xmltoday-delayed-quotes");
    call.setMethodName ("getQuote");
    call.setEncodingStyleURI(encodingStyleURI);
    Vector params = new Vector ();
    params.addElement (new Parameter("symbol",
      String.class, symbol, null));
    call.setParams (params);

    // Make the call.
    Response resp = call.invoke (/* router URL */ url, /*
        actionURI */ "" );

    // Check the response.
    if (resp.generatedFault ()) {
      Fault fault = resp.getFault ();
      System.out.println ("Ouch, the call failed: ");
    } else {
      Parameter result = resp.getReturnValue ();
      System.out.println (result.getValue ());
    }
  }
}
```

Listing 2

```
public synchronized void addObject(Object key, Object
value)
{
if (key != null && value != null)
{
cache.put(key, value);
}
}

public Object fetchObject(Object key)
{
if (key == null)
{
return null;
}
return cache.get(key);
}
```

Listing 3

```
public StockQuote getQuote(String symbol)
{
StockQuote stockQuote = null;
if(symbol != null)
{
stockQuote = (StockQuote) cache.fetchObject(symbol);
if(stockQuote == null)
{
synchronized(cache)
{
   stockQuote = (StockQuote) cache.fetchObject(symbol);
   if(stockQuote != null)
   {
      return(stockQuote);
   }
      Vector params = new Vector();
      params.addElement(new Parameter("symbol",
        String.class,      symbol, null));
      stockQuote = (StockQuote) soap.invoke("getQuote",
         params);
      if(stockQuote != null)
      {
         cache.addObject(stockQuote.getSymbol(), stockQuote);
      }
}
}
}
return stockQuote;
}
```

Download the code at

sys-con.com/webservices

Altova

www.xmlspy.com

Written by Nigel Thomas

Building Asynchronous Applications & Web Services

Java messaging can help you meet the challenge

With the advent of J2EE 1.3, we've become familiar with the message-driven bean (MDB) as a key architecture component, and the use of asynchronous messaging as an aid to application scalability. But there are many more ways in which messaging can be used to build robust Java applications and Web services.

AUTHOR BIO:

Nigel Thomas is the director of product marketing at SpiritSoft. Prior to SpiritSoft, Nigel spent five years with EAI vendor Constellar, serving in consulting, support, sales support, and development positions. He also spent more than eight years at Oracle Corporation architecting and delivering Oracle's accounting products and then moving to worldwide performance consulting and CASE development.

NIGEL.THOMAS@SPIRIT-SOFT.COM

Today, as always, enterprises are faced with the challenges of time-to-market, data distribution, and business flexibility. They are also faced with

- ***Multisite or globally dispersed operations and end users:*** Applications work across the Internet and must be reliable, scalable, and easily manageable.
- ***A rapid and unpredictably changing business environment:*** Architects need to provide for frequent changes during the lifetime of a deployment.
- ***Demanding architectural requirements:*** These requirements typically involve the Internet, the J2EE platform, .NET, Web services, and multiple-legacy execution environments.

How can you use asynchronous Java messaging to meet these challenges? In this article, I'll explore why, when, and how to use asynchronous messaging in Java applications and Web services. I'll also discuss the advantages of an asynchronous model, when to adopt a message-based model, and typical applications of these techniques.

I won't compare and contrast the many different implementation options for Java messaging for a review, see "When Should I Use JMS?" (available at www.sys-con.com/java/article.cfm?id=1275).

A Simple Asynchronous Model

I'm assuming that most readers are somewhat familiar with the basic constructs of a simple asynchronous programming model, including threads, events, and listeners, and the synchronization and locking of shared data.

These constructs are used in multithreaded programs within a single Java VM process. The same basic model also applies with minor changes to C# and .NET – in fact, to any multithreaded model – except the construct names are changed. Java's synchronized keyword is replaced with C#'s lock to control access to critical blocks of code. Essentially, events, listeners, and synchronized data objects and code blocks are today's more flexible and sophisticated versions of earlier languages' interrupts, handlers, semaphores, and mutex constructs.

The asynchronous model can certainly increase throughput and reduce latency (response times) even on single-processor systems, as different threads can simultaneously consume network, I/O, and processor capacity. On multiprocessor machines other related processes (including database shadow processes) can also execute in parallel.

Figure 1 shows some typical interactions between layers in a J2EE architecture. The gold bars highlight periods where synchronous activities may be blocked, waiting for return of information from other layers. If you can shorten the critical path of the overall process by either returning control faster or moving work into the gold sections, you can reduce the overall response time. You can also offload work into longer-running background threads; for example, in principle the Place Order EJB could choose not to wait for the database update to complete before returning control to the servlet.

Program threads interact through shared data (in process, or external) or using passed-in parameters. The more the different threads share data, the tighter the coupling; it becomes much harder to maintain the components sep-

Improv Technologies

www.improv-tech.com/jdj/download

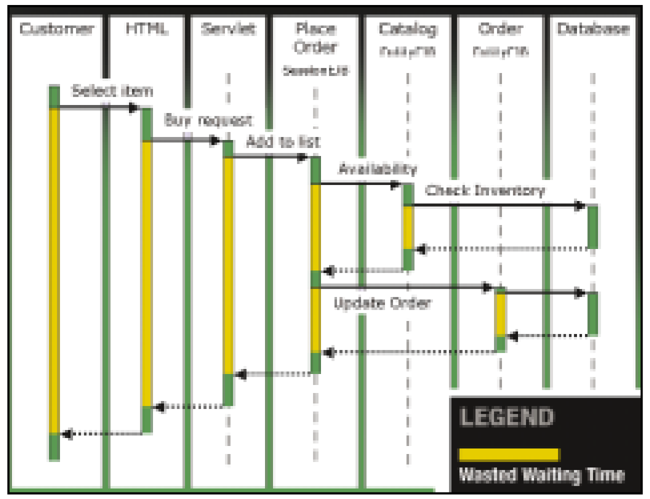


FIGURE 1 **Sequence diagram – placing an order**

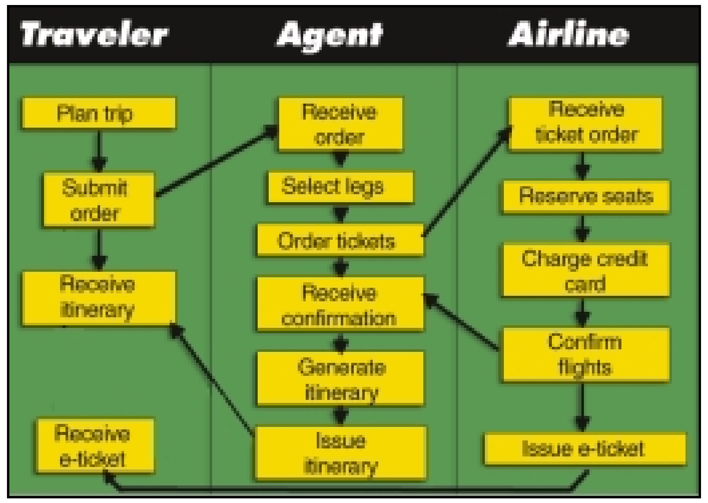


FIGURE 2 **Flight reservation**

arately. Shared data also raises the possibility of data corruption; locks or semaphores (synchronization) must be used to avoid the possibility of two threads simultaneously updating data objects. This complicates the programming model and tends to reduce performance as threads serialize (block and wait) as they queue up for access to locked data items.

When to Adopt Messaging

You've seen some of the advantages of an asynchronous model, and at the same time some of the limitations that a single-process, multithreaded model can impose. Let's look at how moving to a multiprocess, message-based asynchronous model can add significant benefits to your applications.

Document-Centric Processing

Figure 2 shows the most basic reason for executing a business transaction using multiple processes. In this simple example, three actors are involved: the Traveler, the Agent, and the Airline. Each has control over its own activities – but no one is in charge of the whole "conversation." Just as in the real world, the various parties exchange documents (business messages, if you like) to trigger each stage of the overall business transaction. This is because there's no single database they can all share. This style of interaction can be called *document-centric processing* to distinguish it from the database-centric model commonly used to build internal systems such as ERP and CRM. The document (the message passed from system to system) contains all necessary data items and takes the same role as input parameters in a well-structured function call.

Transactional Islands

No actor in a value chain like this can safely make assumptions about specific vendor or technology-platform decisions made by his partners. Each actor is a "transaction island," and interaction has to be loosely coupled.

In document-centric process models (Web services are a prime example), business conversations are characterized by the different, maybe interlocking or interlaced, transaction scopes at work. In this type of system, messaging is used to safely pass control between the different actors without tying together their individual physical transaction islands.

In the flight reservation example, any flow across the lanes (between actors) is implemented as a message. Flows within a lane could also be messages – that depends on the scale of each separate subsystem.

An inevitable consequence of the separate development of these autonomous systems is that system A (the Agent) can't know how long it will take for system B (the Airline) to complete its part of the conversation (reserving seats and billing a credit card) before replying with a flight confirmation message. In these circumstances, a tightly coupled two-phase commit transaction simply isn't feasible – both technology and organizational politics rule it out.

Compensating Transactions

To be able to reverse the effect of any exceptions that may arise, you have to approach errors and corrections using compensating transactions.

Think of using your credit card in a store. You buy a sweater, pay with your credit card, then notice that you've picked up the wrong size. The sales assistant doesn't simply roll back your transaction. Instead, he or she performs a second, compensating transaction that reverses the effect of the mistake. You get both credit card slips, and you should see both the debit and the credit on your monthly statement. Any nontrivial business conversation will anticipate the possibility of errors and need for corrections.

Figure 3, a RosettaNet model of a purchase order, shows how both the buyer and the seller can compensate for errors, production problems, or lack of end-user demand by varying the details of the order over a period of hours, days, weeks, or even months. It also highlights that the purchase order is just one part of a bigger buyer/seller relationship – loosely integrated with other conversations like Quote, Forecast, Shipment, and Billing/Payment.

The Benefits of Messaging

As before, you can benefit from improvements in throughput and latency – but now in a wider range of cases. You can also introduce scalability, as you can load balance not just between threads on a single processor but also across servers in clustered and distributed systems.

Messaging also offers increased reliability and availability. Because the message server itself can be clustered, message consumers (such as J2EE MDBs) can be spread over multiple (redundant) host servers; in the event of a failure of one part of the message-server cluster, message clients can seamlessly reconnect and retransmit any unacknowledged messages.

Finally, it's easy to reconfigure message flows by interfering at the message layer. Messages can be filtered, enriched, transformed, and rerouted using simple standards-based tools such as XSLT; or published on a topic to multiple subscribers. Any message broker can be used to achieve this kind of adapt-

Macromedia

www.macromedia.com

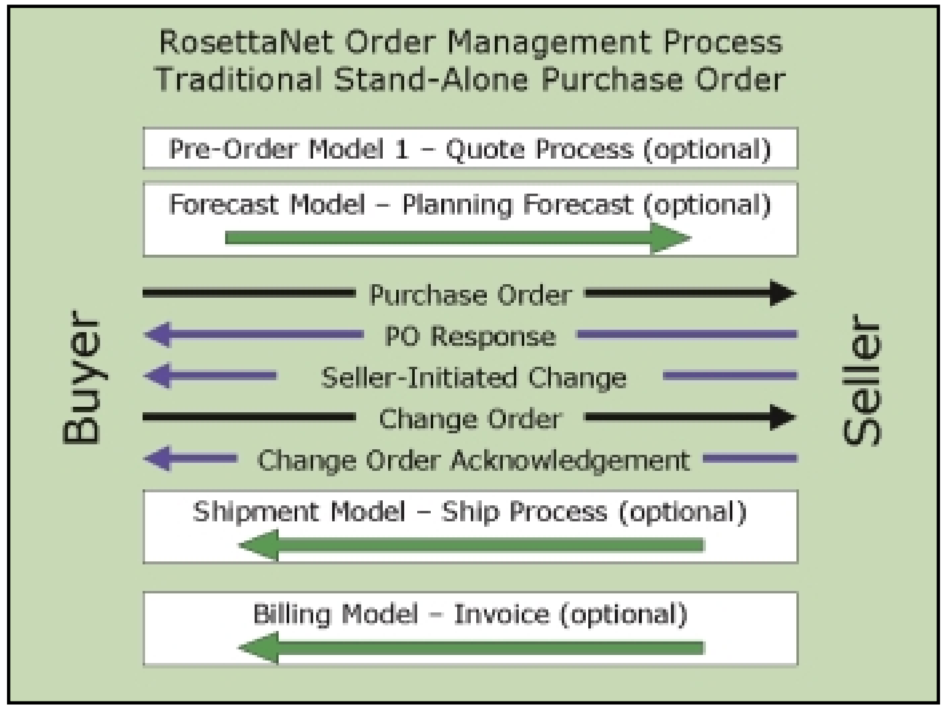

RosettaNet Order Management Process
Traditional Stand-Alone Purchase Order
Pre-Order Model 1 – Quote Process (optional)
Forecast Model – Planning Forecast (optional)
Purchase Order
PO Response
Seller-Initiated Change
Change Order
Change Order Acknowledgement
Shipment Model – Ship Process (optional)
Billing Model – Invoice (optional)
Buyer
Seller

FIGURE 3 | **RosettaNet purchase order**

ability. Try doing that with a remote procedure call!

Separation of Concerns

Perhaps the most important benefit of a document-centric approach is the clean separation of application components – each of which can be owned, developed, and deployed autonomously – either because different actors (as in our flight reservations example) are involved, or simply because the best-of-breed application components used to construct a single business application may use different technology platforms. When you have millions of dollars invested in software assets, the last thing you want is to trash them just because they don't match your current preferred platform.

Different document-centric components can have different development life cycles, languages, and platforms. You can mix-and-match your 30-year-old mainframe systems – using MQSeries to kick off CICS transactions – with your J2EE/UNIX, Windows/.NET, and any other legacy processing.

Using message-oriented middleware, it's easy to plug these components together. Because it's easier, it's also cheaper. Interfaces tend to be simpler and cleaner, which discourages the spaghetti interconnections typical of multithreaded programming and drives toward a true loosly-coupled approach. You can assemble systems from best-of-breed pieces, which may be hosted in different departments or even companies Management of each component is autonomous; each can be separately scaled, replicated, or replaced. Easier integration also promotes more frequent reuse rather than redevelopment of components.

Typical Applications

So, what kind of applications are we talking about in which asynchronous messaging patterns can be applied? System-to-system workflow, which is a catch-all for any kind of fully automated integration; in-house enterprise application integration (EAI); and external integration between businesses – what we used to call B2B, then e-business integration, now called Web services.

The integration may involve multiple deployment platforms – J2EE, .NET, or any number of legacy platforms, hardware, and software, from many different vendors. CIOs want to be able to plan their infrastructure to be completely uniform, but however hard they try, along comes a merger, acquisition, new technology, or just an opinionated CEO to thwart their tidy ideas.

Even if you've managed to stick to J2EE, you're probably looking at more than one vendor's product stack. Maybe you're developing on JBoss but deploying on BEA WebLogic, with a nice helping of IBM WebSphere and SonicMQ on the side, and perhaps a sprinkling of TIBCO Rendezvous as well.

In financial services, message-oriented systems have been in place for the past 10–15 years. Over the past five years Java, J2EE, and JMS have brought increasing productivity benefits as the institutions reach toward straight-through processing and next-day/same-day settlement.

In the telco market, in spite of the sector's deep recession, JMS is a key component of the OSS/J (Operational Support Systems for Java) initiative. A message-based approach is used to simplify the integration of ordering, provisioning, billing, and network management systems. With industry-wide support for messaging and formatting standards equipment manufacturers, service providers and network operators can easily plug in new products and services.

In manufacturing, we're seeing RosettaNet and (more slowly) ebXML evolve to define and coordinate complex supply-chain processes; and now we're beginning to see Web services technology, with SOAP as the transport, adopted to support all kinds of loosely coupled B2B integration.

Process description and choreography standards like BPML (Business Process Markup Language), WSFL (Web Services Flow Language), XLANG (the choreography language used by Microsoft's BizTalk), and WSCI (Web Services Choreography Interface) are being used to represent how individual collaborating processes are tied together by message flows, both in XML and intuitive graphical notations.

Conclusion

It all boils down to this: a message-based, document-centric approach is appropriate for any integration of autonomous components in a business process flow – providing the technology underpinnings you need to connect your functional systems to each other and then to the rest of your organization and your employees, customers, partners, suppliers, and regulators. Most developers today are faced with the need to connect heterogeneous applications and services across the inherently unreliable and unpredictable Internet; they can make it easy on themselves by simply using asynchronous Java messaging.

References

- Thomas, N. "When Should I Use JMS?" *Java Developer's Journal***.** Vol. 7, issue 1.
- Ross-Talbot, G. and Brown, G. "Scalable Web Services using JMS & JCache." *Web Services Journal*. Vol. 2, issue 3.
- Ross-Talbot, S. "Building to Scale." *Java Developer's Journal*. Vol. 7, issue 2.

Altaworks

www.altaworks.com

Internet Operating Systems

MAPPING TRADITIONAL COMPONENTS TO A NEW PLATFORM

It's banal but true. The network, or the Internet, has indeed become a computer. It's possible now to assemble an application from network-based components called Web services.

The network serves as the platform, or "computer," for such an application. Granted, the new platform is not yet suitable for building robust business applications. Developing and maintaining such applications still requires significant effort. In addition, some of the required Web services are not yet fully defined. At the same time, it is clear now where the industry is going. We even have a new term describing the emerging platform – the Internet Operating System (OS), sometimes also called the Grid.

In this article I'll look at traditional computer and operating system components and map them to components of the Internet OS.

CPU

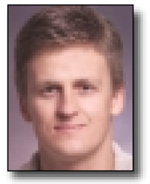
The Internet can be viewed as a massively parallel and distributed multiprocessor. A single task can be broken down into multiple threads of execution and run in parallel on multiple computers in the network. This approach is known as grid computing. In some cases, when there is no, or limited, coordination of the nodes involved, it has been called peer-to-peer computing. Not every application can be effectively parallelized. The ones that can will leverage enormous computing resources distributed over the Internet.

Peer-to-peer applications recently made a lot of press. We all remember Napster, Gnutella, SETI@Home, and numerous others. Some of them are quite scandalous. In one interesting case, a system administrator who installed a grid-computing screen saver on several machines in a university network was charged with stealing the school's network bandwidth and CPU cycles. In spite of cases like this, and in spite of what music industry execs would like us to believe, grid and peer-to-peer computing offer great potential for solving real business problems. Peer-to-peer systems are being studied at MIT, Berkley, Stanford, and Microsoft Research. Sun Microsystems and IBM recently released grid computing–related products and are working actively in this area.

File Storage

File storage is a very important component in any platform, and the Internet OS is not an exception. The file system of the Internet OS consists of the storage devices of the numerous computers connected to the network. According to recent calculations, the Internet has about 500 million users worldwide. If we take this to be the rough estimate of the number of computers connected to the network, and assume that each machine has one gigabyte of spare disk space, we get a file system with a capacity of 5 * 10^{17} bytes, or five hundred thousand petabytes – and this is a very conservative estimate. Even if we partition this disk space among many applications, each application still gets a good chunk of storage – storage that is sufficient for saving many copies of the entire library of recorded music and video, for example. It's not a surprise, therefore, that applications leveraging this resource, such as Napster, Gnutella, Kaaza, and others, have become immensely popular with consumers. Recently, we've seen the emergence of business systems operating on the same principle. One could mention products from companies such as NextPage and Jibe. Support for WebDAV, an XML standard that defines how files and folders can be manipulated over the Internet and in J2EE and .NET platforms, is yet another proof of the Internet file system idea gathering momentum.

Companies have traditionally focused on managing file storage concentrated in data centers or distributed over local area networks. In the near future, we can expect introduction of new products related to management of resources of the Internet file system and implementing functions such as distributed search, indexing, replication, fault-tolerance, and others.



Memory Hierarchy

A memory hierarchy is used to bridge the gap between the speed of the CPU and the speed of the IO system. The memory hierarchy of a traditional computer system consists of the storage subsystem, the main memory, and one or more levels of the CPU cache. Multiprocessor machines can be built with distributed shared memory or memory subspaces that are private to each of the CPU modules.

Author Bio

Dmitri Tcherevik is a technology strategist for Computer Associates in the office of the CTO. After extensive development experience in Ingres and the Jasmine object database, He led the development of CleverPath Enterprise Content Manager and Advantage Integration Server. Before joining CA, Dmitri helped develop a computer chess program under the leadership of Mikhail Botvinik, the renowned world chess champion. He graduated with honors from the Department of Cybernetics at the Moscow Institute of Physics Engineering specializing in distributed systems, databases, and logic programming.

DMITRI.TCHEREVIK@CA.COM

If we consider the Internet as a massively parallel multiprocessor with a vastly distributed file system, then it becomes obvious that it can benefit from a memory hierarchy design of its own. The cache maintained by a Web browser on the local file system is one of the elements in this hierarchy. A Web page can be retrieved from the browser's cache in a matter of milliseconds. The size of this cache is infinitely small, however, compared to the volume of the file storage distributed over the Internet. As a consequence, the probability of a cache miss is very high, and when there is a cache miss, it may take seconds to retrieve the same piece of content from a remote server. In addition, some of the content types, such as video or audio streams, cannot be cached locally and must always be retrieved from a remote server. Obviously, there is a need for an additional level in the memory hierarchy of the Internet platform.

This gap is being filled by content delivery networks (CDN) such as the ones from Akamai, Exodus, or AT&T. A content delivery network caches popular content on edge servers that are located near, in network terms, the end-user machines consuming the content. In addition to CDNs, companies are experimenting with clever peer-to-peer content replication schemes that place content close to where it is used. Instead of playing a movie from a central server, for example, a network node may choose to stream it from one of its nearby peers. Blue Falcon Networks is an example of a company in this space.

The area of content caching, replication, and delivery is still being very actively researched and developed. The near future should bring some interesting products and ideas.

Messaging

Interprocess communication (IPC) and messaging are important components of any platform. Shared memory, pipes, and sockets are examples of low-level IPC mechanisms. They form the foundation for higher-level mechanisms such as JMS, MSMQ, IIOP, SMTP, HTTP, and others. Some of these protocols are universally implemented, such as HTTP. Others tend to be platform or vendor specific, such as MSMQ.

The industry converged on HTTP as the preferred communication mechanism for the new Internet OS. Web-based components composing an Internet application can be deployed on computers with vastly different operating systems. HTTP, in conjunction with other Web standards such as XML and SOAP, ensures that these components can easily exchange data and services. Interoperability of Web services is paramount to the success of the new platform, and several standards bodies are working hard to ensure that interoperability is achieved and then preserved as the platform matures.

Directories

Directories have traditionally been used by operating system and application server components to locate resources and other components. JNDI, for example, is a J2EE directory that is used to register and look up EJBs, RMI servers, JDBC connection pools, and other resources.

A Web-based application must be able to locate two types of resources distributed widely over the Internet: data and services. It can then add value by transforming data with services. The term "data" is somewhat misleading in this context. It's typically used to describe structured information saved in a database. Only a small fraction of information available on the Web matches this description. Most of the information is unstructured and exists in the form of images, documents, audio files, and so on. Therefore, we will follow the industry practice and use the term "content" to denote information resources accessed by Web-based applications. In summary, a Web-based application requires two types of directories: a content directory and a directory of available Web services.

UDDI is one of the fundamental standards of the new Internet OS that define directories of Web services. With the help of this directory, an application can easily locate a service that implements a certain interface. In other words, an application can locate a service based on its functional description. The Internet is vast, and a directory may contain entries for a large number of services that implement identical interfaces. The standard in its current form does not specify how these services can be distinguished based on their nonfunctional characteristics such as latency, price, or mean time to failure. Obviously, some additional work is required in this area.

A single standard describing a universal content directory for the new Internet platform does not yet exist. WebDAV can be used to fill the void when needed, but it does not yet define some of the critical functions, such as content categorization or search. I expect a lot more activity in this space in the near future. Content and services are equally important to a Web-based application.

User Interface

Microsoft Windows, X-Windows, Mac OS X Quartz, Java Swing, KDE, and GNOME are graphical desktop environments that have traditionally been used to develop user interfaces for client/server applications and applications running on a single computer.

While it's possible to build a desktop-based user interface for a Web-based application, they tend to have Web-based user interfaces. This phenomenon is easily explained by the fact that while traditional applications are only available locally or on a LAN, Web-based applications can be accessed from any device, such as a PC, a PDA, or a mobile phone connected to the Internet.

Graphical desktop environments of the client/server world were replaced with portals in the world of Internet-based applications. Just as a desktop environment is capable of aggregating several desktop applications, a Web portal is capable of aggregating user interfaces of multiple Web-based applications. Many of the features found in desktop environments, such as customization, floating windows, drag-and-drop, and others, made their way to portal products. At the same time, many people still prefer desktop environments to Web-based portals for day-to-day tasks such as sending an e-mail or scheduling a meeting. Some work will still be needed to raise the usability and intuitiveness of portal interfaces to that of the desktop environments, and to deliver consistent usability of the interface across many different types of devices.

Security

The list of security services commonly offered by application server and operating system platforms includes user profile management, authentication, authorization, secure communication, and single sign-on to various applications deployed on the platform.

The same list of services will ultimately be offered by the Internet OS to Web-based users and applications. The difference is in the scale. Where systems distributed over a LAN or WAN have to deal with thousands of users, applications deployed on the Web routinely deal with millions of users. In a recent benchmark, CA's CleverPath Portal handled profile information for 2.5 million users, and this number is far from the limit. Public portals such as MSN and Yahoo exhibit even higher levels of scalability.

Single sign-on to Web-based applications and services is critical to ensuring usability of portal interfaces. Microsoft Passport is an example of a single sign-on mechanism used mainly in applications associated with the MSN portal or the Windows platform. Products with similar functionality are available from other vendors and can be used to set up single sign-on domains for Web-based communities of users and applications not related to Microsoft. We can expect a lot more of these products to be offered as services on the Web.

While it is mostly clear now how various aspects of the Internet OS security can be implemented, the standards that would tie together products from different vendors and ensure their interoperability are still largely missing. IBM and Microsoft recently released a standards proposal for Web services security. It's only the first step in a long process.

Transactions

Distributed transaction coordinators became a staple in client/server systems. They can be found in J2EE, COM, CORBA, and other platforms. The purpose of a distributed transaction coordinator is to ensure atomicity, consistency, isolation, and durability (ACID) of multistep transactions spanning multiple systems distributed in a LAN environment.

While ACID properties of transactions are equally important in applications leveraging content and services deployed on the Web, distributed transaction coordinators became largely irrelevant in the context of the Internet platform. They rely on a lot of heavy synchronous network traffic. Communication on the Internet tends to be asynchronous. In addition, latency of network links on the Internet is much higher than that of a LAN, which makes two-phase commit coordination difficult if not impossible. The two-phase commit protocol has not been designed to handle long-running transactions.

The problem of distributed and long-running transaction coordination on the Web is addressed with the help of workflow systems. A multistep transaction spanning several Web services can be represented with a multistep workflow process. A workflow system can use asynchronous messaging to communicate with a remote Web service. All steps in the workflow are atomic and durable and their results are committed immediately. Should any of the steps in the workflow fail, a sequence of compensating steps can be applied to undo changes that have already committed.

A workflow system can be used to coordinate processes involving multiple Web services. These processes can in turn be exposed as higher level services. For example, a trip-planning service can be built as a process coordinating hotel, air, and car reservation services. Many people believe that workflow systems will become the main programming tools for the Internet platform.

Monitoring and Management

Management and monitoring of client/server applications can be done at the application or infrastructure level. In fact, the vast majority of today's applications are managed at the infrastructure level. Instead of managing the program components that implement the business logic, we are managing networks, databases, and application servers that are used to run these components.

This approach cannot be applied to applications deployed on the Internet platform. The infrastructure that is used to run such applications is not only extremely diverse and distributed, but is also controlled by many different and unrelated companies. One can hardly imagine a database agent installed at company X reporting its state to an enterprise management solution like CA's Unicenter node installed at company Y. The situation is complicated by the fact that in the environment where many of the services are replicated and where applications perform late binding to services, it's difficult to outline the bounds of the infrastructure used by an application at any given moment.

These considerations are raising a slew of interesting problems, many of which remain largely unaddressed. The ideas exchanged in the industry revolve around self-organizing networks, management of Web services at the interface level, service level agreements, service level management, and other topics.

Summary

While it's clear that the Internet will become the platform for the next generation of applications, many aspects of this platform still require significant research and development. In particular, the coming months will see a lot of activity and new products in such areas as:

- Grid computing and peer-to-peer systems applications
- Distributed and replicated file system management
- Content caching, replication, and distribution; content delivery networks
- Secure and reliable messaging based on HTTP, SOAP, and XML
- Directories of distributed and replicated content and services
- Ubiquitous user interfaces aggregating Web applications
- Security of Web services, user profile management, and single sign-on
- Management of distributed processes and orchestration of services
- Monitoring and management of applications based on Web content and services

PolarLake

www.polarlake.com

Written by Ron Ben-Natan

Web Services and Wireless Messaging

Innovations in field workforce management

Field workforce management is an application segment responsible for scheduling resources working in the field, assigning work orders, dispatching work, and letting workers report from a mobile terminal. Among those using such systems are utilities, construction crews, maintenance organizations, telecommunication operators, and equipment manufacturers responding to trouble reports. The new generation of field workforce management is based on an application server architecture and allows the field workforce to work in a disconnected mode, which is mandated by lack of full wireless coverage and by the fact that wireless networks are slow and unstable.

AUTHOR BIO:
Ron Ben-Natan, CTO of ViryaNet, Inc., holds a PhD in computer science in the field of distributed computing and has been architecting and developing distributed applications for more than 15 years. Ron's hobby is writing about how technology is used to solve real problems; he has authored numerous books, including IBM WebSphere Application Server: The Complete Reference (Osborne/McGraw-Hil).
RBENNATA@HOTMAIL.COM

Utilizing Web services in this environment usually focuses on integrating workforce management systems with other systems, such as the call center system, the part management system, the financial system, and more. Web services enables faster integration and supports real-time interfacing with a lower total cost of ownership (TCO). This article, however, focuses on the use of Web services on the front end – the messaging between the mobile terminals and the central server – serving business scenarios in which the field workforce communicates with the central dispatch station as well as among themselves.

Because the workforce doesn't always have full connectivity to central dispatch, and because the network bandwidth may be limited and/or expensive, interaction between the field workforce and the server is based on store-and-forward messaging rather than "normal" Web transactions. Because the Web services model is inherently asynchronous, it fits the interaction model between the field and the central server. By ensuring that the data content is formatted using Web services and delivered over wireless messaging technology, the service enterprise can achieve lower TCO not only on the back end but also in field interaction.

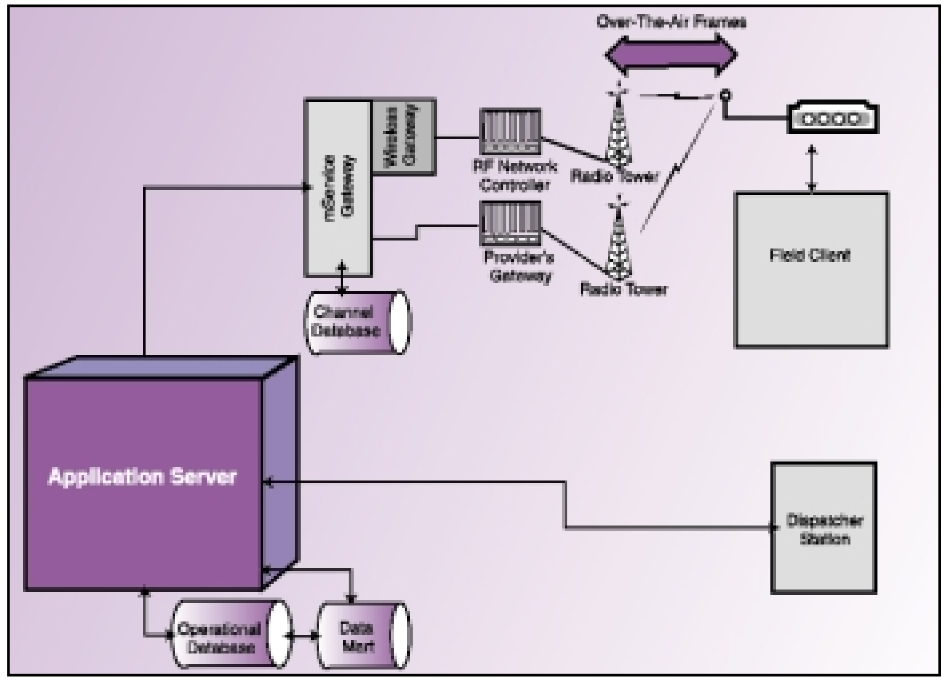


FIGURE 1 Workforce management architecture

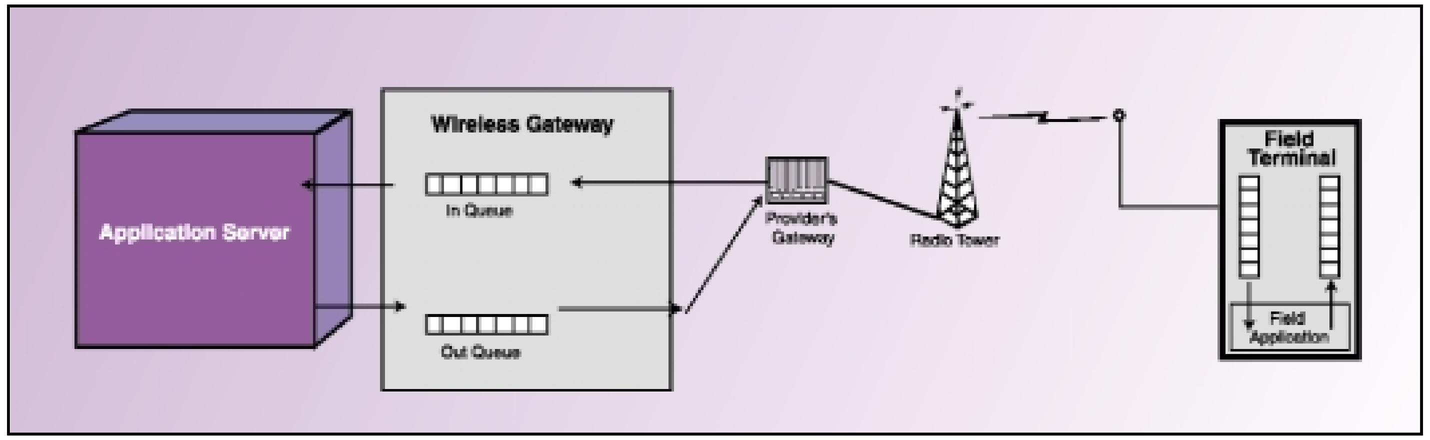


FIGURE 2 Message queues in the wireless gateway

Field Workforce Management

A field workforce management system can be broadly classified in two parts: the scheduling and dispatch servers and the field terminals. As Figure 1 shows, an application server manages all work orders and all interaction with the server, both from the dispatch stations and from the field. The system is accessed over either the LAN/WAN (e.g., the dispatchers) or wireless networks (e.g., field technicians).

A typical scenario involves a work order that comes in from an external system (a call center system, help desk, customer information system, etc.). If back-end integration utilizes Web services, the order is created by invoking a Web service implemented by code running within the application server.

The scheduling module assigns the optimal resource to perform the work, which is then dispatched over the wireless network. This is either pushed out to the client (using a wireless messaging service) or pulled by the client device. Sometimes the implementation involves both – a notification is pushed out to the client device informing it that new data exists, and the data is pulled by the client in response to this notification. In all three cases (push, pull, and push-pull), all interaction is asynchronous because individual components in the system (and especially the mobile terminals) may not be online at a given time.

The asynchronous Web services model is perfect for these environments. While most of the current infrastructure used in these environments is based on proprietary messaging, infrastructure services are now moving to a Web services model in which the message content is packaged as SOAP messages and delivered to an endpoint defined using WSDL.

Wireless Messaging

Most of today's wireless gateways implement wireless messaging using a queuing abstraction. They implement a set of APIs that support message delivery by placing application messages into queues. Messages are received by registering to receive messages through an incoming queue. This is true for both the client and the server, as shown in Figure 2.

A messaging metaphor (also called store-and-forward) is simple to use and the APIs are very clean. The wireless gateway hides all details about the underlying network and will behave in the same way regardless of whether the network is IP-based, whether it does or does not support sessions, whether it requires a connection to be set up or is always on, whether multiple networks should be tried, and so on.

The major advance in the marriage of wireless messaging and Web services is Web services–aware message routing. All wireless middleware supports routing, allowing messages to be routed to the right location based on the application, the message type, or an address. Routing can be very intelligent. Advanced routing features can include routing different message types on different networks. For example, in utilities, maps and schematics may be required in the field; delivering these documents over a slow wireless network isn't feasible. However, when a truck enters a yard it usually has access to an 820.11 network through which it can receive large quantities of data.

Such routing has been based on proprietary methods and formats. A new generation of wireless middleware is on the horizon – one that uses native SOAP routing information. In such systems, the content of the SOAP envelope is inspected to determine where the SOAP request needs to be delivered and to handle additional delivery attributes packaged within the SOAP request.

Independent Contractors

While Web services can be used for all field interaction with the workforce management system, they really shine in the independent contractor scenario. Many companies employ independent contractors, which allows them to utilize skilled workers without staffing up, manage periods with peak load, and avoid complex employment contracts. The independent contractor often works with many service companies and needs to be able to perform work assigned by various sources. The field application used by the independent contractor needs to receive work orders from multiple dispatch centers. Using Web services is probably the only way to do this without incurring very large costs.

Let's look at a business environment in which "Joe" specializes in complex troubleshooting of problems related to large oil

FIGURE 3 Joe's workday viewed using his mobile unit

furnaces. Joe is an independent contractor and may receive work from any number of companies that provide warranty services on such furnaces.

During his workday Joe receives allocated work orders from multiple companies and is in the field doing repairs. Each of these service companies has a scheduling and dispatch system they use to pick the best resource for a given problem. These applications use a Web services–enabled Web calendaring system to access Joe's calendar information. If they select Joe as the optimal resource, they schedule the job using the Web calendaring service and use a Web notification service to inform Joe. Joe then uses his mobile device to access the company's online server through a set of Web services and download the work assigned to him (see Figure 3). Upon completion, another set of Web services closes the work order and uploads information about the finished work.

Peer-to-Peer Dispatching

Web services–aware wireless middleware can provide increased benefits in peer-to-peer dispatching scenarios as well. Normally, all dispatching is done from the central dispatch site by a dispatcher. However, advanced service organizations want the ability to have a field technician or a field supervisor dispatch work to a peer. This can be a transfer of work, a request for assistance, or an employee who is both a worker and a manager performing multiple tasks from the same mobile terminal.

Peer-to-peer dispatching allows mobile field technicians to remotely reassign work orders over a wireless network without the aid of a dispatcher. The new resource can accept, reject, or reassign the order. This capability allows organizations to improve productivity and customer service. The ability to share work in real time results in better workload balancing and increased field utilization. In addition, dispatchers are freed of daily work-monitoring responsibilities, which results in more time allocated to productive planning activities and reduced dispatcher costs.

The key to efficient peer-to-peer dispatching is how quickly the message can be routed to the correct field terminal. In the simplest case, every message has to make it all the way to the application server and be handled by the workforce management system – the entire system is overloaded unnecessarily. A better approach is one in which the wireless gateway can inspect the SOAP envelope and reroute the message directly to another mobile terminal, as shown in Figure 4. In this case, the central server doesn't need to be involved in the dispatch process and needs to be updated only when the work has been accepted by the receiving field technician. Perhaps one day (not for a while yet) one field terminal will directly invoke a Web service hosted by another mobile terminal.

Conclusion

Wireless environments are very different from classic Web environments. The networks are slow, coverage is less than perfect, and the bandwidth is extremely limited. Most importantly, the wireless-messaging world is based on asynchronous store-and-forward. Historically, wireless environments have grown separately from Web technologies, which are mostly synchronous. This is changing, however, due in part to Web services.

Wireless environments have always been message based. All wireless middleware vendors take the wireless-messaging approach, and the facilities they provide involve message queuing. Web environments, on the other hand, have been inherently synchronous and online. A convergence of these two environments is now possible due to the asynchronous nature of Web services. This convergence and the ability to run Web services over wireless middleware hold much promise for companies that have had to put up with high TCO because their environments have been highly proprietary. However, this technology is still very young – it remains to be seen whether it can deliver on the promise.

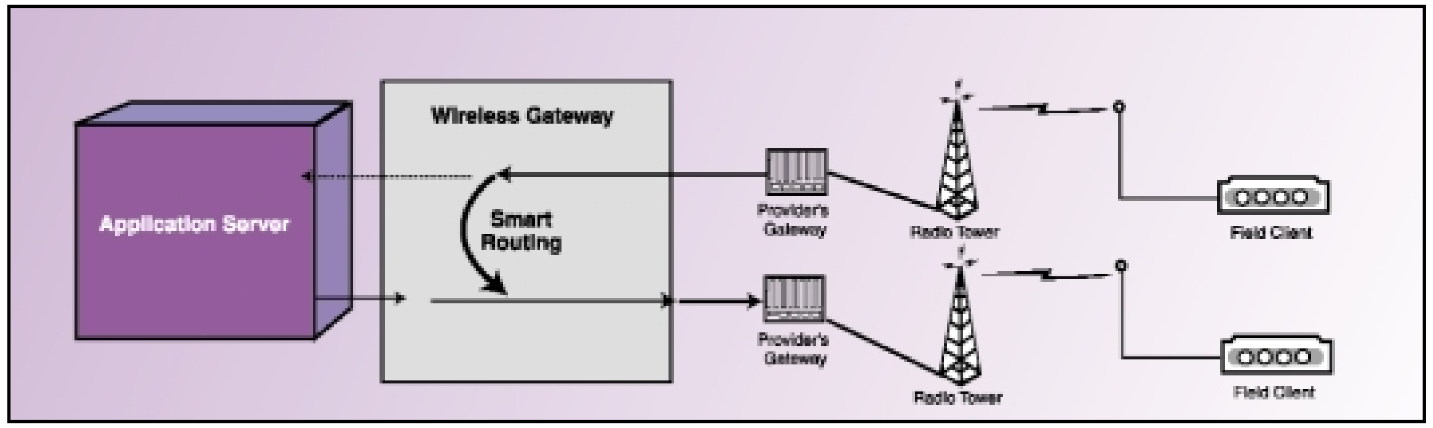
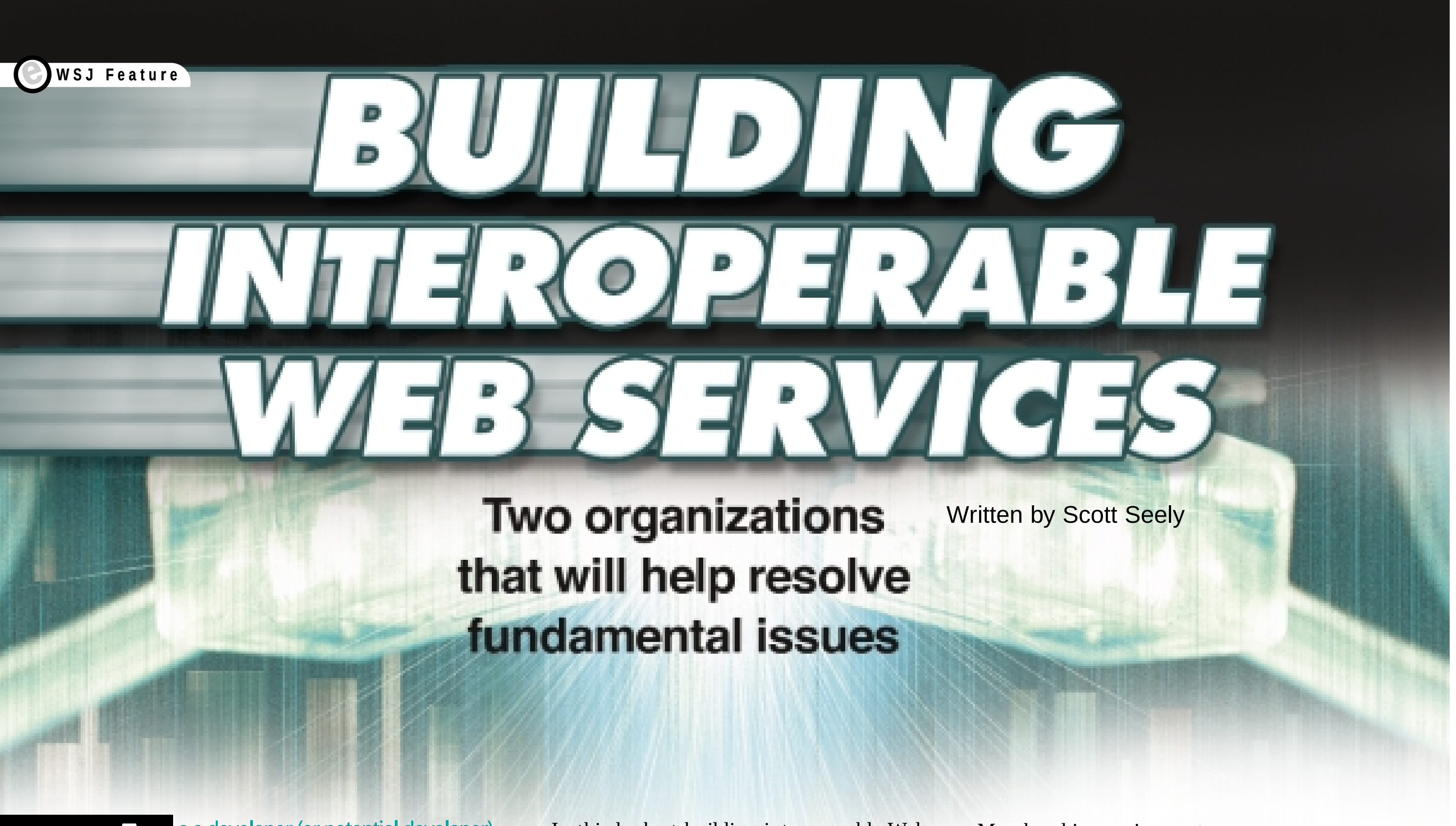


FIGURE 4 Efficiently routing the peer-to-peer dispatch

PolarLake

www.polarlake.com



BUILDING INTEROPERABLE WEB SERVICES

Two organizations that will help resolve fundamental issues

Written by Scott Seely

As a developer (or potential developer) of Web services, sooner or later you are going to wonder just how you might be able to reduce the interoperability issues you see. If you work with Apache Axis or Apache SOAP 2.x, you will see this problem when sending a java.lang.Hashtable to a Microsoft technology-based Web service. Similarly, creators of Microsoft ASP.NET Web services express confusion when sending a System.Data.DataSet to a non-Microsoft endpoint. The bad news here is that the developer of the Web service may have actually thought they were creating an interoperable solution.

AUTHOR BIO:

Scott Seely is a member of the MSDN Architectural Samples team. Besides his work there, Scott is the author of SOAP: Cross Platform Web Service Development Using XML *(Prentice Hall PTR) and the lead author for* Creating and Consuming Web Services in Visual Basic *(Addison-Wesley).*

SSEELY@MICROSOFT.COM

In this look at building interoperable Web services, the following items will be covered:

- ***Interoperability efforts:*** Understand what work is being done to make your job as a developer easier
- ***Best practices:*** Understand what you can do as a Web service creator to make it easier to use Web services for more end users

Interoperability Efforts

If you were an early adopter of Web services, you may have had problems getting two different SOAP stacks to talk to each other. You could have seen the problem with something as simple as a Web method that echoed back “Hello World!” Fortunately, that kind of experience is rapidly becoming a thing of the past. Why?

Today, there are two large efforts underway to ensure that the various vendors of SOAP toolkits are creating interoperable implementations. Web services stacks have little value if they can only interchange data between machines hosting the same SOAP stack. This was true before in the HTTP world and continues to be true today with SOAP. The two efforts are SOAPBuilders and the Web Services Interoperability Organization. Both groups exist to get the various implementations to work with each other. Their methods of achieving this goal are significantly different. They differ in many areas:

- Membership requirements
- Stated goals
- Community makeup
- Progress to date

In this section, we will take a look at these two complementary organizations and see how they, together, are going about making application integration through Web services possible. It's important to remember that many of the organizations represented in SOAPBuilders are also represented in WS-I.

SOAPBuilders

SOAPBuilders came into being January 30, 2001. A lot of the discussion that this group now hosts was hosted originally on the SOAP discussion list at http://discuss.develop.com/soap.html. You can read the announcement of the new newsgroup here: http://discuss.develop.com/archives/wa.exe?A2=ind0102&L=SOAP&P=R506. This newsgroup exists because Tony Hong of XMethods actually acted on the idea of creating a new newsgroup. Why does this newsgroup exist? From the original message, Tony states, “The group was started to facilitate communication between builders of SOAP implementations, specifically about interoperability issues.” Since then, the group has done quite a bit to make interoperability a reality. What have they done?

First off, three suites of interoperability tests have been defined. These tests can be thought of as progressively higher bars. The first set makes sure that basic data types can be exchanged without any issues. The second set makes sure that SOAP Headers as well as arrays and complex types can be exchanged by the SOAP stacks. These two sets of tests all use rpc/encoded message exchanges. The latest, third round of testing makes sure that document/literal encoding is understood. The tests also verify WSDL compatibility for the various SOAP stacks and helper tools.

So, how does a member of the SOAPBuilders community find out where other endpoints live? One way is through the mailing list at soapbuilders@yahoogroups.com or by searching the archives at http://groups.yahoo.com/group/soapbuilders. Another is by looking at a page showing a list of endpoints:

- www.pocketsoap.com/registration
- www.whitemesa.com/interop.htm
- www.whitemesa.net/r3/interop3.html

The other thing the SOAPBuilders community does is get together for face-to-face events. These events are for developers only. Politicking is kept to a minimum. During these events, the developers representing the various SOAP stacks discuss what things to test next, discuss the specifications to arrive at a common understanding, and test the SOAP stacks in close proximity. This often results in significant headway taking place. In the February 2002 face-to-face, the Axis toolkit “learned” how to understand document/literal WSDL and messages. In the June 2002 face-to-face, the group discussed whether or not it would make sense to start adding programming concepts to the common vernacular of SOAP. One such example is dictionary types. A dictionary type is one that contains unique keys to lookup values. In Java, this is called a hashtable. In C++ and the STL, it is called std::map. In .NET, it's called a dictionary. No specifications exist (yet) to explain how one SOAP toolkit should represent a dictionary-data type. Still, this is something that the SOAPBuilders community is pondering.

In order to join the group and attend a SOAPBuilders face-to-face event, you should be involved in the development of a SOAP stack or set of SOAP tools. The only real membership requirements are a live endpoint that the development team is willing to host. To attend a face-to-face event, potential attendees need to do the following:

- RSVP the host of the event.
- Pay for transportation to the event.
- Bring a laptop, preferably with an 802.11b card. Wireless makes networking everybody much easier.

Why does all of this matter? As a Web service developer, I've found that using SOAP stacks that are actively involved in the SOAPBuilders community reduces interoperability pains. I have also discovered that it is very important to only use stacks that were released after the development team writing the stack got involved in SOAPBuilders. No involvement in SOAPBuilders typically means that I won't even bother with the stack.

And, in case you think SOAPBuilders is completely formal, the other reason for the face-to-face events is so that the community can get to meet. This way the people get to know each other. Oh, and it's really hard to be mean to a person you've had a few beers with.

Web Services Interoperability Organization

For all of its merits, SOAPBuilders is still an ad hoc organization with little formal backing. Also, it does not have obvious points of entry for system integrators, consulting firms, and others who use SOAP but do not create SOAP stacks. The Web Services Interoperability Organization (WS-I) was created to solve the interoperability problem in a different, more structured manner. The organization is creating profiles, sample applications, and testing tools for Web services. Let's take a quick look at these items.

A profile defines a collection of protocols and Web standards. A profile definition explains the types of solutions that combination of protocols can provide. The first profile that will come out of WS-I is called the Basic profile, located at www.ws-i.org/docs/WS-IProfiles.doc. The Basic profile essentially defines what it means to be a Web service. This profile includes the following specifications and standards:

- XML Schema 1.0
- SOAP 1.1
- WSDL 1.1
- UDDI 1.0

The profile will provide guidance about how to use the specifications. Things such as useful subsets, usage guidelines, and other information will appear in the profile documentation. Using the profile, WS-I will also produce a set of sample applications. The first set of sample applications will make use of SOAP over HTTP. Also, a set of testing tools and test suites are being created. The test tools consist of a message logger and an analyzer. The message logger captures messages and stores them in a message log. The analyzer reads the log and verifies that the messages conform to the specifications defined for the sample applications and the profile. These testing tools will be used by developers and customers to verify the correctness of implementations of the Web services sample applications.

To join WS-I, the potential member must sign a membership agreement and pay an annual membership fee. Members of WS-I will help to define and/or refine various standards around Web services.

Best Practices for Building Interoperable Web Services

How can you build an interoperable Web service? Most SOAP stacks allow a great deal of flexibility in sending and transmitting SOAP messages. Because of this flexibility, it is very easy for a developer to make some decisions that make sense for their preferred SOAP stack that wind up being difficult to use when going to another implementation. To increase the likelihood of getting a Web service to work with others, the developer needs to look at three primary things:

- SOAP stack being used
- XML Schema (XSD) describing the messages
- Usage of toolkit specific features

By being aware of all of these items when developing a Web service, the developer can spend more time coding and less time supporting end users.

Choosing a SOAP Stack

When you look for a SOAP stack, make sure that the one you choose to work with meets these requirements:

- The development team participates in some form of cross–Web service stack testing.
- The development team has live endpoints for all rounds of testing.
- The SOAP stack interoperates with other active endpoints.

By following this advice, you will find that a number of simple interoperability problems will not surface. Surprisingly, this is the only nonintuitive thing to look for. From the list of participating options, choose one that supports the language and platform you prefer to work with. If you don't find a compatible toolkit, either pick a new language or prepare to develop one. While developing the new toolkit, make sure you join in with a community that does testing across Web service stacks. SOAPBuilders does this today. Other groups, such as WS-I, will have more advanced forms of interop testing in the near future.

While not a requirement, you will find that interoperability will be at its best if the particular SOAP toolkit has some form of representation in WS-I.org. The representation indicates that the organization behind the Web service implementation wants the interoperability to go deeper than things like "can the toolkit read and send a complex type." WS-I.org will look at the complete scenario: discovery, proxy generation, message exchange, etc., and make sure the toolkit can do it all.

Choosing a Data Representation

A good number of interoperability problems happen due to developer choices, not SOAP toolkits. Here is an example of a developer choice that results in a WSDL-type definition that is hard to use:

```
<xsd:complexType>
    <xsd:sequence>
       <xsd:element ref="xsd:schema" />
        <xsd:any />
    </xsd:sequence>
</xsd:complexType>
```

When a client sees this definition, what should they do? It appears that the WSDL has stated something along the lines of "I will send you an XSD and an XML document that corresponds to that XSD." (That wasn't explicitly stated, but that's what this snippet is trying to express.) How do I code against this? Is this code simple or complex? What do I do? Here is another example of a hard-to-use WSDL definition:

```
<xsd:complexType name="Map">
    <xsd:sequence>
        <xsd:element name="item"
minOccurs="0" maxOccurs="unbounded">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element
name="key" type="xsd:anyType" />
                    <xsd:element
name="value" type="xsd:anyType" />
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
```

Once again, I know that I am getting two items called key and value that are of type anyType. My guess as a developer is that this data is a dictionary datatype. What should I map the key and value elements to when I receive the data? I haven't got any contextual information.

In case you don't recognize the above snippets, the first one is the result of telling ASP.NET to return a System.Data.DataSet. The second one is an Apache SOAP serialization representation of java.lang.Hashtable. Instead of returning one of these non-interoperable types, you could choose to return something that is more likely to interop. For example, when returning tabular data, it is fairly unlikely that the number and names of the elements will vary based on inputs to the Web method. Most of the time, the tabular data will contain the same types. Instead of returning a generic type, the WSDL and corresponding source code could be modified to return something that other toolkits can more easily import and use. How could something like the ASP.NET example be improved?

Let's assume that the DataSet is populated with product data. I could define the array of data with the schema shown in Listing 1.

Now, any consumers of the data know exactly what the response SOAP will look like. Instead of needing to process the returned message with XML, the client program could use native classes. Most SOAP toolkits are capable of translating between the XML and object representation of data. Well-defined WSDL allows the toolkit to generate these objects with no to little developer intervention.

You can get similar problems when just stuffing XML inside of the SOAP Header or Body. Unless the XML is really generic (and it rarely is), take the time to define what the WSDL will look like. Often, this choice is made when the developer knows things will be changing and they don't want to update the WSDL in order to change the code providing the Web service. Any time you change the way a Web service behaves, you will probably break a downstream client somewhere. Take a little extra time and define what the returned XML will look like. When you update the back-end Web service, plan some time in the schedule to write a new version of the WSDL.

Often, when using SOAP, the developer knows that they want to handle messages as raw XML. That's fine. Still, the returned XML will typically follow a set schema. Define that schema in your WSDL, even if you don't rely on object serialization/deserialization. Just don't take away the ability to generate smart clients from any end users. By providing schema, you make the end user's life much easier.

This last bit of advice invariably brings up the question: what should I do with two WSDL files? The issue: most people want to avoid having too many side-by-side versions of a Web service up and running. One solution is to have a regular release cycle and make sure that users of the Web service understand that the service will only be available for a fixed amount of time. For example, put documentation in the WSDL that states when the service endpoint will end service. If the Web service requires a preexisting arrangement (such as a signed contract) before a user begins accessing the service, put language in the contract explaining how frequently the Web service will be updated and how long any given version of the Web service will be available. This still means that you will wind up running multiple versions of the same Web service at the same time. Thankfully, it gives you a way of limiting how long versions need to be available.

Finally, since you are using WSDL and schema anyway, you might as well code the Web service to emit a document/literal representation of the data instead of rpc/encoded. There are situations where rpc/encoded still shines. Namely, if you are serializing a complex object graph and the relationships need to be maintained, use rpc/encoded. Otherwise, stick with document/literal representations of data.

Summary

To achieve Web service interoperability, the tools vendors have banded together and created two organizations: SOAP Builders and WS-I.org. SOAPBuilders is an informal group that works at getting fundamental issues around interoperability resolved. At the other end of the spectrum, WS-I.org provides a more formal structure that rigidly defines what it means to be interoperable and the set of protocols that are being worked on. WS-I.org provides architectural guidance, testing tools, and sample applications that aid in the understanding and adoption of Web services. We need both organizations to solve the issues around interoperability and Web service adoption and general understanding. Why? Some implementations of SOAP stacks, such as SOAP::Lite (Perl) and pocketSOAP (COM/PocketPC), may never have corporate backing. Implementations like these are valuable and need some forum in which to verify interoperability with the Web service stacks offered by corporate entities.

As a developer of Web services, you should take advantage of the work the interoperability groups are doing by only using SOAP toolkits that are actively involved in SOAPBuilders. It makes your life easier because you are using a toolkit that is being tested against other toolkits. Issues are discovered and fixed through this testing.

When developing a Web service, avoid declaring the return of generic XML in WSDL files. Instead, declare exactly what you are returning via schema and return that instead. Is generic XML useful in interop scenarios? Yes. But most scenarios do not require or benefit from the ability to send and read weakly expressed XML. Unless you really need that ability, express the format of the returned message in XSD contained or referenced from the WSDL description of the Web service.

Listing 1

```
<xsd:complexType name="ArrayOfProduct">
    <xsd:sequence>
        <xsd:element minOccurs="0"
            maxOccurs="unbounded"
            name="Product" nillable="true"
            type="s0:Product" />
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Product">
    <xsd:sequence>
        <xsd:element minOccurs="0" maxOccurs="1"
            name="QuantityPerUnit" type="xsd:string" />
        <xsd:element minOccurs="1" maxOccurs="1"
            name="UnitPrice" type="xsd:decimal" />
        <xsd:element minOccurs="0" maxOccurs="1"
            name="CategoryName" type="xsd:string" />
        <xsd:element minOccurs="0" maxOccurs="1"
            name="SupplierName" type="xsd:string" />
        <xsd:element minOccurs="0" maxOccurs="1"
            name="Name" type="xsd:string" />
        <xsd:element minOccurs="1" maxOccurs="1"
            name="ID" type="xsd:long" />
    </xsd:sequence>
</xsd:complexType>
```

Download the code at
sys-con.com/webservices

Reviewed by Joseph A. Mitchko

Actional SOAPswitch

Web-enable your enterprise today

About the Author:

Joe Mitchko is a lead engineer working for Nekema, Inc., a leading Web-service technology provider for the insurance industry sector. Joe is also the product review editor of Web Services Journal.

JOE@SYS-CON.COM

Actional

Web Services Journal MAGAZINE WSJ World Class PRODUCT Award

GLOBAL HEADQUARTERS
701 N. Shoreline Blvd.
Mountain View, CA 94043
Tel: 650-254-4100
Web: www.actional.com
E-mail: sales@actional.com

EVALUATION DOWNLOAD
http://www.actional.com/downloads/software/index.asp

LICENSING INFORMATION
SOAPswitch pricing is dependent on project and deployment specifics; for typical enterprise deployments, ASP is $250,000.

TESTING ENVIRONMENT
OS: Windows-XP
Hardware: Dell Inspiron 8000

I must admit that the subtitle I used here sounds a bit like an overused marketing cliché broadcast from a late-night commercial on television. Anyone seriously involved in enabling enterprise resources as SOAP-based services on the Web knows that the "today" part is quite a stretch. Although, depending on how you look at it, you may be able to take it quite literally.

For instance, suppose you were able to take software assets that currently exist behind the corporate firewall and plug in a network gateway or router that has the ability to instantly render them as Web services. Imagine that all you need to do is define which back-end components you want exposed as Web services, specify various network and security permission settings, open the firewall gates, do some UDDI advertising, and you're ready to go. And, imagine that this can be done across a variety of custom and packaged systems within the corporate enterprise using a simple administrative interface. As you might have already guessed, the scenario I just described is available today with SOAPswitch from Actional Corporation.

Overview

SOAPswitch is a nonintrusive proxy server that exposes custom and packaged applications as Web services without any programming involved. It's designed to be situated behind the corporate firewall with other back-end systems, allowing you to manage all of your Web services under one point of control. It can route existing Web services on the back end, or serve as a proxy for J2EE or COM custom applications and a variety of leading packaged systems such as SAP and Siebel. You can consider it as more to a Web service gateway or router than as an application server.

SOAPswitch is a direct descendant of Actional Control Broker 3.0, with a lot of new Web services-based functionality added. This is especially important when you take the various packaged software adapters into consideration for robustness and reliability. You're not talking about a 1.0 release here.

Product Application

SOAPswitch is designed for those organizations which have a variety of custom and packaged applications across a multitude of platforms and technologies that want to combine these systems under a centralized Web services–delivery platform. Integrating these systems into a corporate Web-services substrate does not require developers to code each Web service, but instead is a matter of configuring and deploying each back-end system interface that SOAPswitch can see. The product does not contain any data transformation or mapping tools, nor does it contain any business process management tools. Therefore, don't expect it to manage sophisticated transactions across disparate systems. That will require a considerable amount of analysis and development, and an application server with Web-service capabilities. But, SOAPswitch can take the various systems in your legacy environment and expose the various application interfaces under a common Web-service substrate – the first step in achieving more advanced Web services for your enterprise.

Features

Some of the key features of SOAPswitch include:

- Adapters to custom application platforms, including J2EE and COM
- Adapters for leading packaged software products, including SAP, PeopleSoft, and others
- Performance monitoring and reporting
- Event logging and reporting
- Web-service auditing
- Security management (including support for single-login authentication servers)
- System alerts (will send an e-mail if a particular service is having trouble)
- Security profiles

First Impressions

Installing SOAPswitch on my Windows 2000 Server–based system was smooth and effortless. The only problem was that there was no indication that the SOAPswitch Windows service was up and running when the install completed. I guess I've been trained over the years to expect some sort of kick-start operation to start things off. Once I figured out that SOAPswitch was running, I quickly moved on to the browser-based administration console to configure the server. I found the setup wizard to be clear, unambiguous, and extremely well documented, and one of the best examples of a browser-based wizard I've worked with in a while (see Figure 1).

One neat feature of the configuration process was when it came time to set up the alert e-mail. After entering the necessary configuration information including SMTP server, e-mail address, etc., the system will deliver an email confirmation message to you. Quite accidentally, I entered my POP3 address instead of the SMTP server. The configuration wizard clearly reported the e-mail setup error and gave sufficient information to diagnose the problem. In all, it took me 10–15 minutes to set things up, although if I hadn't messed things up setting up the e-mail, it would have taken less time.

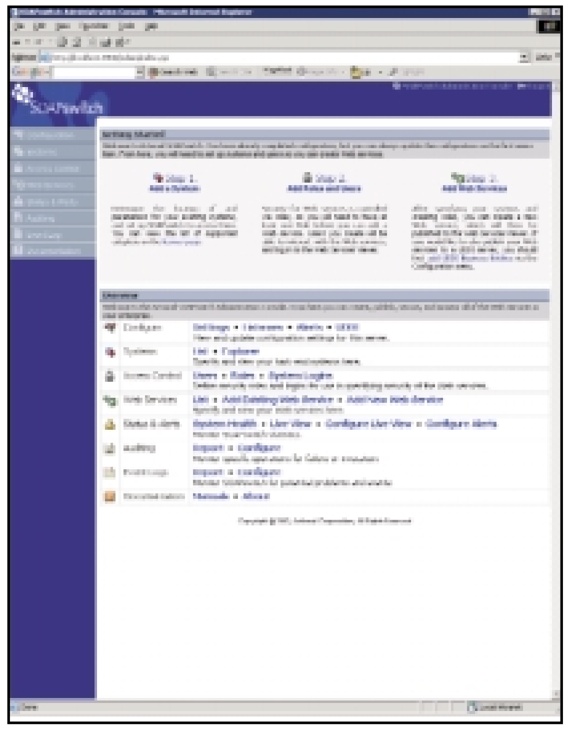


FIGURE 1 SOAPswitch Admin Console

Making my way further through the administrator console, I was quickly able to take an existing Web service using the WSDL URL and have SOAPswitch act as a proxy. In addition, SOAPswitch was able to introspect the various classes and EJB components managed by a WebLogic server I had previously installed on the test system. Configuring a Web service from an EJB component was equally straightforward (see Figure 2).

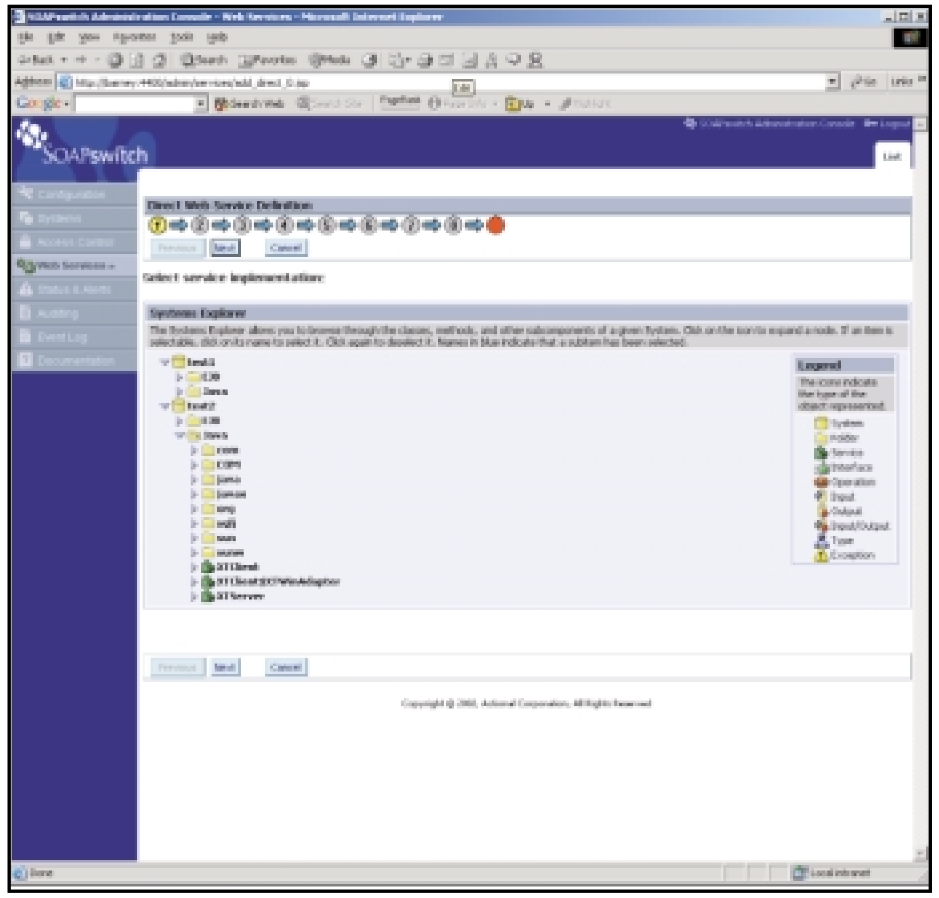
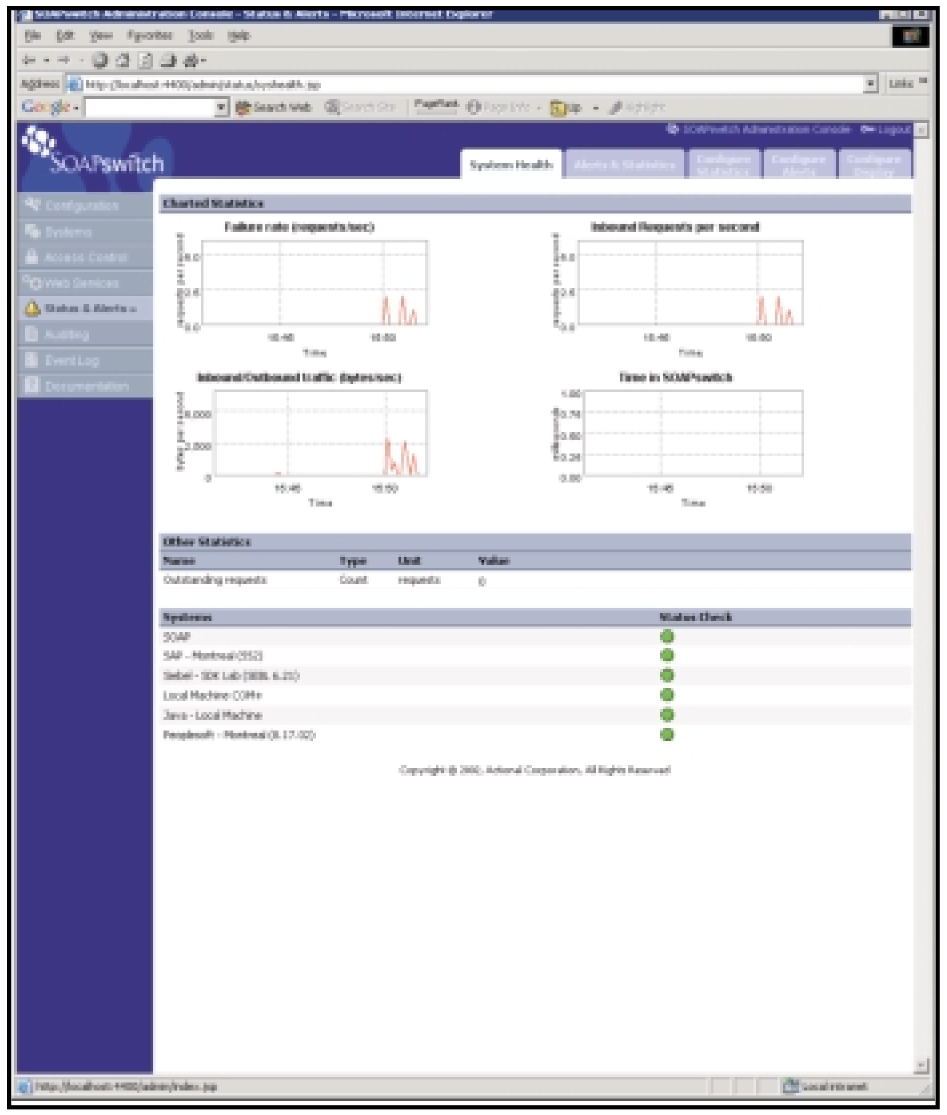

SOAPswitch Administration Console - Web Services - Microsoft Internet Explorer
SOAPswitch
Direct Web Service Definition
Select service implementation
Systems Explorer
Legend

FIGURE 2 Systems Explorer in the Add Service Wizard

One note: in order to browse WLS components, you will need to manually configure a CLASSPATH property value in SOAPswitch so that it points to the WLS deployment directory area. If I could improve one area, it would be the ability to view EJB components remotely using the WLS T3 protocol instead of having to directly reference the JAR files on the SOAPswitch server.

Testing

Testing a Web service using SOAPswitch is limited to being able to view and manually edit the request and response SOAP messages using the product's Web Services Viewer. More detailed testing will require you to use some other process or tool. An alternative approach would involve downloading a Java client stub


SOAPswitch Administration Console - Status & Alerts - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:4400/admin/status/syshealth.jsp
Links
SOAPswitch
SOAPswitch Administration Console
Logout
System Health
Alerts & Statistics
Configure Statistics
Configure Alerts
Configure Registry
Configuration
Systems
Access Control
Web Services
Status & Alerts
Auditing
Event Log
Documentation
Charted Statistics
Failure rate (requests/sec)
Inbound Requests per second
Inbound/Outbound traffic (bytes/sec)
Time in SOAPswitch
Time
15:45
15:50
http://localhost:4400/admin/index.jsp
Local intranet

Other Statistics

Name	Type	Unit	Value
Outstanding requests	Count	requests	0

Systems	Status Check
SOAP	🟢
SAP - Montreal (552)	🟢
Siebel - SDK Lab (SEBL 6.21)	🟢
Local Machine COM+	🟢
Java - Local Machine	🟢
Peoplesoft - Montreal (8.17.02)	🟢

Copyright © 2002, Actional Corporation, All Rights Reserved

FIGURE 3 Charted statistics in the Admin Console

produced by SOAPswitch and using that as a basis for a custom test bench application for your service.

Security

SOAPswitch has built-in support for the HTTPS protocol and provides certificate and keystore management tools. You can also configure SOAPswitch to authenticate users through a directory server or from a single-point authentication system such as Netegrity SiteMinder. With the rich set of security features provided with SOAPswitch, you can configure a Web service to use the HTTPS protocol from an existing non-authenticated service on the back end or map a single authentication ID to multiple back-end accounts. There is also support for setting up security profiles.

Status and Alerts

One of the main selling points of SOAPswitch is its ability to perform centralized monitoring, logging, and reporting for all Web-service activity. A system health tab in the SOAPswitch administrator provides you with real-time status (using the familiar stoplight green/yellow/red indicators) for each of the configured back-end systems. You also get a set of four real-time charts providing you with inbound and outbound requests, failure rates, and other vital statistics (see Figure 3).

Another important feature is the ability to set up custom alerts for a specific Web service and condition. For instance, I can have the system notify the administrator via e-mail if the legacy service takes too long to process a particular service, or if it fails for any reason. The ability to quickly identify problems occurring in the back end cannot be stressed enough in a Web-service production environment.

SOAPswitch provides rich auditing and report capabilities for all Web-service activity. Persisted by a back-end relational database system, you can audit specific services and report on any number of metrics, both real-time and historical. This will be very useful for tracking down those pesky back-end problems.

Event Logging

SOAPswitch maintains an event log that captures all error and warning conditions across the system. Like the auditing feature, you can also configure custom reports based on logging information. In addition, you have control over which severity levels are being logged by the system.

Product Availability

SOAPswitch ships in two versions: Professional Edition (no high-availability features and can only connect to J2EE-and COM-based services) and Enterprise Edition (support for packaged software solutions, high availability, and clustering). Actional SOAPswitch is currently available directly from Actional. Additional Web services products will be available from the company soon.

Summary

SOAPswitch is one of the first products to adopt a new and exciting concept working its way through the Web Services Interoperability Organization – that the various protocols involved in Web services (SOAP, WSDL, UDDI) can potentially be integrated with the application layer of the OSI model (known as layer 7 markup). This makes perfect sense, and relegates SOAPswitch into a new product class that views Web-service integration from a network perspective – hence the name "SOAPswitch".

UNDERSTANDING THE WEB SERVICES VISION
EVALUATING and ADOPTING WEB SERVICES
A pragmatist's view
Scalable Web Services using JMS & JCACHE
BETTING YOUR FUTURE: .NET OR J2EE?
WebServices JOURNAL
.NET or J2EE

WSJ INTERVIEW

Grids, Peers, Discovery, and What's a GAIA?

A conversation with Graham Glass

AUTHOR BIO:

Michael Sick is an independent Java architect helping clients solve complex product definition and design problems. He has more than eight years of experience in the construction of distributed information systems and Internet technology, holding positions including architect and VP of development. He holds undergraduate degrees in both geology and political science from Guilford College.

MIKE_A_SICK@YAHOO.COM

This month *WSJ* focuses on P2P architectures and grid computing, two topics that are gaining momentum in our industry. Over the past year or so I've read many excellent articles and books on these topics. However, getting a handle on what P2P and the grid are can be a challenge as implementations advance rapidly, major technologies are converging, and more people are applying these concepts to their particular disciplines.

I recently had the opportunity to interview Graham Glass, CEO and founder of The Mind Electric. We discussed the definitions of P2P and grid computing, the importance of service discovery, how TME's products relate to P2P/grid concepts, and the future of service-oriented computing. I've known Graham for some time and have always found his work and insight to be both solid and valuable. This article should give both developers and managers a better perspective on the state of the technology and how grid and P2P can help them solve problems.

Graham, tell us a little bit about yourself and The Mind Electric.

Graham: I'm the CEO and founder of The Mind Electric. TME builds software infrastructure for creating systems out of Web services, specifically moving our clients toward service-oriented architectures. Additionally, we believe that there will be a strong convergence between technologies like Web services and architectures like P2P.

What were you doing prior to starting The Mind Electric, and how did you get involved with distributed computing?

Graham: I've actually been in distributed computing for a long time. In a previous lifetime I founded a company called ObjectSpace, which had a good reputation for a product called Voyager. Voyager was in the days when object request brokers were kind of state of the art. The goal of Voyager was to make distributed computing based on object request brokers as simple and easy as possible. That theme of "making things as simple as possible" is something I've always held near and dear to my heart. The Mind Electric is doing the same kind of thing, but for much more sophisticated software infrastructure problems. We're in the business of making the construction of professional-quality enterprise systems out of Web services as simple and easy as possible.

We're hearing the term peer-to-peer, or P2P, frequently in the press. From your point of view, what is peer-to-peer computing?

Graham: I think from a technical perspective, when people talk about peer-to-peer they're really talking about highly decentralized architectures. For example, a client/server architecture is one where the server is generally a beefy machine with a whole bunch of mission-critical stuff and the clients are dumb. That's a setup that is clearly not a P2P system because the server is way more powerful and resilient than other nodes. I would say personally that peer-to-peer is more of a mind-set.

It's much like the days of object technology, where a lot of people didn't really know what an object was. They would ask for an example of objects in practice. I have an example that I've used quite successfully to show people what P2P is really all about; it has to do with cell phones.

If you look at how cell phones work right now, your phone is quite dumb. To set up a cellular network, you have to install base stations at a number of different points. This is quite like the client/server model – the phone is the client and the base station is a fairly extensive server. To extend the cellular network you have to put a bunch of base stations in place.

In a peer-to-peer cellular network, the cell phones would be both client and server. So if a cell phone were not being used as a way to communicate, then it would automatically switch into a kind of transmission mode and be able to route other people's cellular phone signals. Now, without needing base stations, you could basically air drop 100,000 cell phones into someplace and boom! – Instant cell network!

Another hot topic is "grid computing." Could you speak about that and its relation to peer-to-peer computing?

Graham: Grid computing is something that was initially about how to use free machine cycles to do high-speed scientific calculations. But as people get into Web services they're saying, "If I create a Web service, when I publish it for use by other services, what is the technology that will allow those services to connect together?" In other words, what will connect the producer and the consumer? And if a Web service fails, what will automatically route requests to a new server? The industry is realizing that a good architecture for doing this work is very similar to the architecture for managing services in the domain of electricity distribution. In the national power grid, if you're a producer – not of a Web service but of electricity – you can easily plug in a power generator somewhere. If some consumer wants to use electricity, they effectively plug into the national grid and it is responsible for connecting producer and consumer, billing, routing, and load balancing.

In the realm of electricity it's very clear what "the grid" does; people are now saying, "Hang on, we can do the same thing, but for Web services and for XML data." People are looking at grids as a general-purpose concept for linking together producers and consumers of services and data. It just so happens that when you start building large-scale grids, it's very natural to use P2P architectures to implement them because P2P architectures can usually scale much better than client/server architectures.

The Mind Electric's two products are GLUE and GAIA. GLUE is a core Web services platform. Can you describe GLUE and then tell us a little about GAIA and how it is a P2P and grid-computing enabler?

Graham: Our product line is designed to help enterprises go through the adoption curve of Web services from beginning to end. The beginning is, "How do I build my first Web service?" Developers then progress towards the mental evolution of "How do I now build systems comprised of a large number of services hosted in a variety of different platforms?" So the first phase of adoption is what GLUE is targeted at. For Java developers it's all about how do I rapidly create and then deploy Web services. The very early versions of GLUE simply focused on the early parts of the puzzle, but the latest versions include advanced features like a high-speed Web server, servlet engine, JSP implementation, and all the standard J2EE application stuff. The main reason GLUE is so successful is because it makes this very simple and that's what we think Java developers really want.

We think that once people start building Web services, some of which will be built using Java, some built using .NET, then in order to connect these services together reliably you're going to need something that is the equivalent of the national electricity grid. GAIA is targeted at tackling this second stage of evolution. If you imagine having a bunch of different services created in a vendor-neutral way, then GAIA is designed so you can plug something into it and, just like the national electricity grid, it will connect the producers and consumers to perform fail-over, load balancing, discovery, etc. But most importantly, GAIA does this all in a way that's independent of the service implementations themselves.

Service discovery has been an issue for a couple of platforms, JINI and UDDI being two of the most prominent. Please speak about them and GAIA's service discovery.

Graham: Let's look at JINI to begin with. JINI, at least so far, hasn't been particularly successful and one of the reasons was because it was so Java-centric. In a world that is so heterogeneous, with services written in C, Perl, and C#, having a discovery mechanism that's based purely in Java is never going to be successful.

However, one of the things that people liked about JINI, especially in the Java world, is that it was relatively transparent in the way that you used it. So if there was a Java interface for a currency exchange service, it was very easy to say, "Find me a service that implements this interface" and you would get back a proxy and invoke the service transparently.

Java developers using GAIA will find it even easier than any code examples you might have seen in JINI to discover and use services. With one line of code in GAIA you can say, "Find me a service that has a certain interface," and it will find that service if it's compatible, regardless of whether the actual service is an EJB, C#, or VB component – it makes no difference to GAIA. So the first difference is basically ease of use, which we've got a great reputation for.

As far as UDDI goes, it's really just an API to perform a search using XML. UDDI doesn't actually enforce any particular architecture for doing that. One of the things we're looking at is providing a UDDI skin over GAIA. In other words, if you want to access GAIA as if it was a UDDI server, then you can do it through the standard UDDI API. Under the hood GAIA would be using a P2P architecture to actually broker services and the publications.

You said that one of the weaknesses of JINI was that it was Java only. Do you think the JINI team at Sun has learned that lesson and is applying it to JXTA?

Graham: Good question. The lesson they learned most clearly shows up in JXTA just because it's now completely focused on protocols not implementations. I think that JXTA has definitely learned a lot from the JINI experience. It's tricky to understand right now where JXTA is going. It seems to be absorbing more things and becoming less and less JXTA-like over time. Additionally, I think when people look at the architecture in GAIA they'll find it to be insightful.

Do you have any plans for implementing GAIA in languages other than Java? Is there a need to do that?

Graham: It's funny you should ask that. A lot of people who are familiar with The Mind Electric and GLUE would pigeonhole us as a Java-centric company. That's certainly our roots, and we really love Java as a language, but the reality is that there are going to be a lot of other technologies out there, specifically Microsoft's .NET. We wanted to make GAIA as efficient as possible for the platforms that it's actually installed on. Also, we want GAIA to run reliably and quickly on small devices like PDAs, and a lot of PDAs look like they're going to start shipping with the .NET Compact Framework. Later on this year we're going to be porting GAIA to the C# language so it will be a native .NET implementation of the same technology. There will be a .NET version for .NET developers and a Java version for Java developers. They will all completely interoperate. So if you want to build a grid composed of services using .NET it's completely transparent as far as GAIA goes. It uses XML for all of its communications; there's nothing language-specific about the architecture at all.

Back to the grid. When UDDI came out there was a lot of talk about big public UDDI registries. The reality has been that that model has not taken off and it is being downplayed, often at the expense of private UDDI registries. Right now the talk is about 'the grid'; do you see a single grid emerging or do you see many grids?

Graham: I think there'll be a global grid that will share general information and services, but companies will also have their own internal grids. Your company, even within projects, might have mini-grids. The mini-grids are almost like next-generation app servers, and they might be connecting 10 or so different computers. A small grid will presumably be able to access some of the power of the large grid that it's plugged into and so on. Public services will be used by private grids but not the other way around. That's my prediction.

We have it on record. Thank you and best of luck with the upcoming releases of GLUE and GAIA.

The editorial boards of *Web Services Journal* congratulate the recipients of the first annual

EDITOR'S CHOICE AWARDS

WEB SERVICES JOURNAL WSJ EDITOR'S CHOICE AWARD

Web Services Application Server
Winner: BEA Weblogic Server 6.1 (BEA Systems)
First Runner-up: .NET Server (Microsoft)
Second Runner-up: IBM WebSphere Application Server 4.1 (IBM)

Web Services Automation Tool
Winner: Actional SOAPSwitch (Actional)
First Runner-up: BEA WebLogic Server 6.1 (BEA Systems)
Second Runner-up: Oracle 9*i* JDeveloper (Oracle)

Web Services Book
Winner: Building Web Services with Java (Sams Publishing)
First Runner-up: Java Web Services (O'Reilly & Associates)
Second Runner-up: Professional XML Web Services (Wrox)

Web Services Class Library
Winner: CLX: Component Library for Cross-Platform (Borland)
First Runner-up: .NET (Microsoft)
Second Runner-up: SilverStream eXtend Director 3.0 (Silverstream/Novell)

Web Services Foundation Platform
Winner: .NET Framework (Microsoft)
First Runner-up: BEA WebLogic (BEA Systems)
Second Runner-up: SonicXQ 1.0 (Sonic Software)

Web Services Framework
Winner: .NET Framework (Microsoft)
First Runner-up: Oracle 9i JDeveloper (Oracle)
Second Runner-up: Systinet WASP (Systinet)

Web Services IDE
Winner: Visual Studio (Microsoft)
Furst Runner-up: BEA WebLogic Workshop (BEA Systems)
Second Runner-up: Delphi 6 (Borland)

Integrated Services Environment
Winner: IBM WebSphere Studio (IBM)
First Runner-up: BEA WebLogic (BEA Systems)
Second Runner-up: XML Spy Suite 4.2 (Altova, Inc)

Web Services Integration Tool
Winner: webMethodsIntegration Platform (webMethods)
First Runner-up: SOAPswitch (Actional)
Second Runner-up: WebSphere MQ Integrator (IBM)

Web Services Legacy Adapter
Winner: IBM CICS Gateway (IBM)
Runner-up: Actional SOAPSwitch (Actional)
Second Runner-up: WebLogic Integration (BEA Systems)

Mass Market Web Service
Winner: SOAPswitch (Actional)
Runner-up: IBM UDDI Business Registry (IBM)
Second Runner-up: dataLive (Long Reach Software)

Web Services Middleware
Winner: BEA WebLogic (BEA)
Runner-up: GLUE (The Mind Electric)
Second Runner-up: Sonic XQ 1.0 (Sonic Software)

Web Services Platform
Winner: .Net (Microsoft)
First Runner-up: WebSphere (IBM)
Second Runner-up: Oracle 9*i* AS and Oracle 9*i* JDeveloper (Oracle)

Web Service Site
Winner: OASIS
First Runner-up: Xmethods (Xmethods)
Second Runner-up: IBM develoerWorks (IBM)

Web Services Testing Tool
Winner: XML Spy Suite 4.2 (Altova)
First Runner-up: IBM AlphaWorks Web Services Testing Area (IBM)
Second Runner-up: Oracle 9i JDeveloper (Oracle)

Web Services Training Tool or Program
Winner: Web Services ToolKit (IBM)
First Runner-up: RADical Web Services for e-Business (Borland)
Second Runner-up: The Middleware Company

Web Services Utility
Winner: SOAP Toolkit 2.0 (Microsoft)
First Runner-up: SOAPswitch (Actional)
Second Runner-up: Oracle 9i JDeveloper (Oracle)

The awards will be presented at the Web Services Edge Conference & Expo, October 1-3, San Jose, CA

Web Services Java XML .NET Wireless

web services EDGE conference&expo | JDJ EDGE conference&expo | XML EDGE conference&expo | wireless EDGE conference&expo

The Largest Web Services, Java, XML, .NET and Wireless Conference and Expo

WWW.SYS-CON.COM
NEW LOWER REGISTRATION
THE BEST CONFERENCE BUY OF THE YEAR!

Focus on Web Services
Those companies that get an early jump on Web services to integrate applications for the enterprise will be the winners in today's challenging landscape. Take three days to jump start your Web services education!

Focus on Java
Hear from the leading minds in Java how this essential technology offers robust solutions to *i*-technology professionals and senior IT strategy decision-makers.

Focus on XML
Here's your chance to hear up-to-the-minute developments in standards, interoperability, content management, and today's new quest for more efficient and cost-effective Internet and intranet-enabled business process integration.

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

CONFERENCE: OCTOBER 1-3, 2002
EXPO: OCTOBER 2-3, 2002
SAN JOSE McENERY CONVENTION CENTER, SAN JOSE, CA

Reaching Beyond the Enterprise

The Conference...
Organized into five comprehensive tracks:
Web Services, Java, XML, Wireless, and IT Strategy
- Over 60 Information-packed Sessions, Case Studies and tutorials
- Thought leaders and innovators presenting the topics you most want to hear

Free...
FREE Web Services Tutorial to the First 100 to Register.
Attend a FREE Web Services Tutorial on October 3.
Register by August 15th to reserve your seat!

Who Should Attend...
- Develpers, Programmers, Engineers
- *i*-Technology Professionals
- Senior Business Management
- C Level Executives
- Analysts, Consultants

The Largest Expo...
Over 75 exhibitors will deliver their best product demonstrations, stand ready to hear your challenges, and help you find solutions. Don't miss this important opportunity to expand your options.

PLATINUM SPONSOR
IONA

GOLD SPONSORS
ADOS | bea | ORACLE | VIGNETTE

SILVER SPONSOR
Novell

CORPORATE SPONSORS
COREL | Sun | www.xmlspy.com | Rational the e-development company

MEDIA SPONSORS

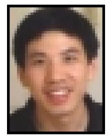
Next-Generation Web Services Discovery

The evolution of e-business integration

Written by Liang-Jie Zhang

Exploring a business application published as a Web service in the UDDI registry or Web Services-Inspection (WS-Inspection) documents is a critical issue. A search for such an application should be effective in terms of time and uniform in terms of interfaces. This article will examine the existing Web services publishing and corresponding discovery technologies, including potential problems and a novel technology for advanced Web services discovery.

AUTHOR BIO:



Dr. Liang-Jie Zhang is a researcher at IBM T.J. Watson Research Center. He has more than 12 years of experience in creating novel technologies and products for e-business integration, streaming media, and intelligent information appliances. Liang-Jie is actively working on enhanced Web services technologies for B2B integration. He has numerous patents and is chairing "Web Services Computing" at IC 2002 and other conferences.

ZHANGLJ@US.IBM.COM

Web Services Publishing

The emergence of Web services represents the evolution of e-business. Web services are reusable Web components with standard interfaces described in WSDLs. They can be accessed by universal clients such as wireless devices, Web clients, and other application clients. As an enabling technology, they've been adopted to represent services and communicate with other services in a standard way.

In general, Web services can be published to a public or private UDDI registry or Web Services Inspection Language (WSIL) documents, illustrated in Figure 1. The design of UDDI allows simplified searching and allows trading partners to publish data about themselves and their advertised Web services for categorization. A UDDI registry is a central place to store the information and pointers about a Web service. There are two types of UDDI registries: private and public. As an application developer, you can publish your Web services to the existing public UDDI registries operated by IBM, Microsoft, HP, and SAP. If you want to publish your private or confidential Web services, however, you should use a private UDDI registry. For testing or small-scale integration purposes you can publish your Web services to WS-Inspection documents, which enable Web services discovery, deployment, and invocation without the need for a UDDI registry. WS-Inspection documents provide a means for aggregating references to preexisting service description documents authored in any number of formats. These inspection documents are then made available at the point-of-offering for the service as well as through references that may be placed within a content medium such as HTML. For example, there is a URL for developers to locate the WS-Inspection documents on a Web site (e.g., www.be4ws-wsd.com/inspection. wsil).

The UDDI registries and the WS-Inspection documents are tightly associated by WS-Inspection data tag "wsiluddi." Thus, in a WS-Inspection document, you can use a reference pointer to connect to a business or service published in a UDDI registry.

Simple UDDI Search and WS-Inspection Screening

Many developers believe Web services will open up a world of interlocking business services that find each other over the Internet, thus integrating disparate code into useful programs. If this vision is to be realized, users must be able to find services on the vast public network.

There are currently solutions for finding Web services from centralized UDDI registries or distributed WS-Inspection documents published on regular Web servers. As shown in Figure 2, a WSIL document can have a link to point to another WSIL document, and a UDDI registry can be private or public. Think of WS-Inspection as a cheap solution for organizations with existing relationships to discover each others' Web services. UDDI, on the other hand, is more like a traditional Yellow Pages directory that organizations can use to get a series of listings for Web services under different categories. UDDI is the building block that will enable businesses to quickly, easily, and dynamically find and transact with one another via their preferred applications.

As for simple WS-Inspection crawling, a search tool can be easily built on top of a WSIL parser for searching WS-Inspection documents. For example, the Web Services Inspection Language for Java API (WSIL4J) provides a Java interface that can be used to parse existing WS-Inspection documents or programmatically create new ones. The WSIL4J is included in the Web Services Toolkit (WSTK) release (www.alphaworks.ibm.com/tech/webservicestoolkit). Most of the WSIL4J classes represent elements that can appear in a WS-Inspection document. The WSIL4J API also provides a *WSILProxy* class that can be used to easily access certain types of information within a WS-Inspection document. The proxy interface will read the WS-Inspection document and then allow you to directly access the WSDL documents for UDDI business services you need.

As for simple UDDI search, a UDDI client can be used to search a UDDI registry. The typical Java-based UDDI client is UDDI4J (UDDI for Java, www.uddi4j.org), an open-source project initialized by IBM. UDDI4J is a Java class library that provides an API to interact with a UDDI registry. The UDDI Project is a comprehensive, open industry initiative enabling businesses to discover each other and define how they interact over the Internet and share information in a global registry architecture. Currently, UDDI4J version 2 beta is available. It provides full support of the UDDI v2 specification, support for multiple SOAP transports including AXIS (SOAP version 3), debug logging, and configuration capabilities. HP, IBM, and SAP announced in December 2001 that they've committed to supporting UDDI4J.

As shown in Figure 2, a Web services search requester can use WSIL4J to look into WS-Inspection documents represented as WSIL. If a requester wants to search a UDDI registry, he/she needs to use a UDDI client such as UDD4J. As the public UDDI operators, IBM, Microsoft, SAP, and HP provide their own Web browser interface to allow users to specify the search criteria for a specific search from their own UDDI registry.

Problems

In general, UDDI can locate businesses whose identities are well known so users can find out what services they offer and how to interface with them electronically. But one of the big concerns is accuracy. When a specific category is registered along with UDDI registration data, only people searching for that exact category will find the results (see www.uddi.org). Moreover, the simple UDDI search is focused on a single search criterion such as business name, business location, business category, or service type by name, business identifier, and discovery URL. With a projected near-term population of several hundred thousand to a million distinct entities, it's unlikely that searching for businesses that satisfy a particular set of criteria will yield a manageable result set. The current advanced UDDI search is trying to deal with multiple search criteria included in a single search request for a fixed UDDI registry.

Current search APIs are rudimentary at best, and developers must write a lot of code to find the Web services they desire. Especially from an e-business application developer's point of view, it's necessary to send a few sequential or programmed search commands to the UDDI registry or other Web services sources, such as WSIL documents, for information aggregation. The information sources may include multiple UDDI registries and other searchable sources. Obviously, there is a need to provide an advanced search mechanism for Web services to dramatically extend the current search capability, which is based on categories or key words, by improving efficiency and enhancing performance.

The UDDI Browser provided by soapclient.com allows you to search custom or private UDDI registries. One UDDI registry should be specified for each request. This doesn't support the aggregation of search results from multiple UDDI registries or other Web services sources.

Next-Generation Web Services Discovery Mechanism

The script-based search agent will likely play an important role in Web services discovery for Web browser–based Web services,

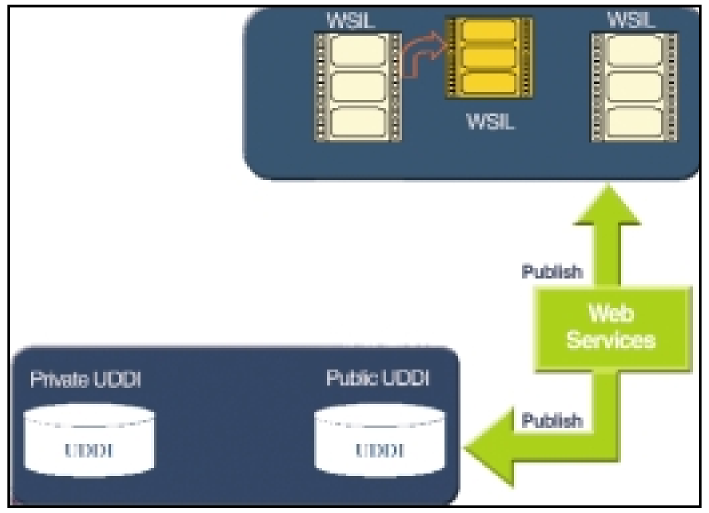


FIGURE 1 | Web services publishing process

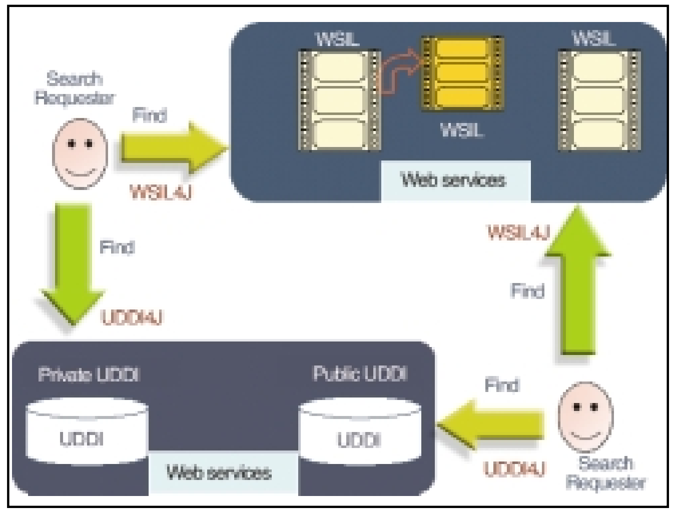


FIGURE 2 | Web services discovery

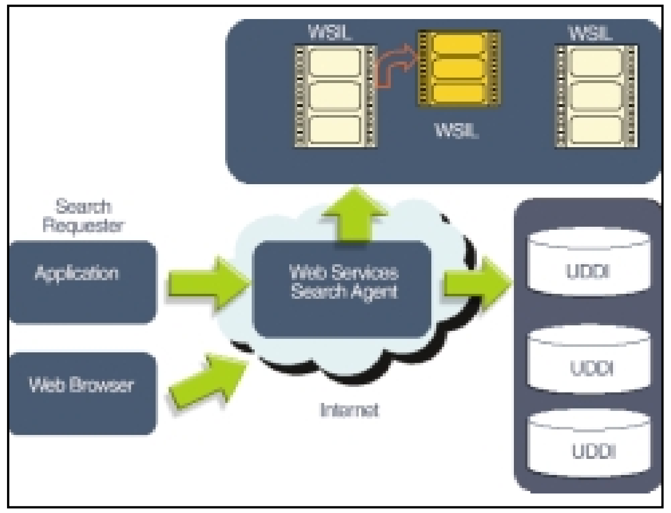


FIGURE 3 Agent-based advanced Web services directory

discovery clients, and e-business applications. The mechanism should:

- Use a standard interface
- Simplify the developer's work
- Hide the complexity of UDDI and WSIL search clients
- Perform result aggregation from one or multiple sources
- Act as an advanced search portal on an application server

Based on these concerns, the next-generation Web services discovery mechanism should look like the search agent shown in Figure 3. The advanced Web services search agent can be accessed by a regular application client or Web browser and communicates with multiple UDDI registries and WS-Inspection documents. A complicated result aggregation mechanism can be defined in the search agent. When a service requester looks for a Web service, three basic data types can be returned from UDDI registries: businessEntity, businessService, and ServiceType (Technical Model, t-Model).

The example aggregation includes, but isn't limited to, intersection-, union-, and script-based logic operation for the resulting responses from multiple sources. The final response to the search requester may be a new XML format or an existing XML format such as WSIL. Due to its extensibility WSIL will play an important role in representing the aggregated result.

In addition, the semantic information about Web services will contribute to efficient Web services discovery. For example, Web services discovery described in DAML for Web Services (DAML-S) involves the automatic location of Web services that provide a particular service and adhere to requested constraints. DAML-S is trying to provide declarative advertisements of service properties and capabilities that can be used for automatic service discovery. The disadvantage of DAML-S is in trying to create a new Web services registry from scratch. If it can describe the semantic information of regular Web services published in UDDI registries or WS-Inspection documents, DAML-S can be adopted by the next-generation Web services discovery mechanism.

On the other hand, the Web Services Relationships Language (WSRL) describes the relationships of the Web services rather than the requests. Current Web services specifications and UDDI specifications lack definitions and descriptions of the generic relationships among business entities, business services, and operations. But these relationships are keys to composing and executing dynamic business processes integration. Web services relationships are defined at different levels: business-business relationships, business-service relationships, service-service relationships, business-operation relationships, service-operation relationships, and operation-operation relationships. The resulting relationships are captured by WSRL. For example, the partner relationship among business entities defined in WSRL can help the Web services discovery mechanism effectively locate desired service providers based on the specific search criteria. Currently, a simple relationship (publisherAssertion) between two parties has been defined in UDDI v2. The WSRL will be a natural extension to publisherAssertion.

The next-generation Web services discovery mechanism should be built on uniform script-based XML search requests a, federated UDDI search engine with result aggregation, semantic information utilization, and relationship definition in WSRL.

An Example: Business Explorer for Web Services

IBM has been pioneering the advanced Web services discovery work through its alphaWorks technology – Business Explorer for

WebServices .NET J2EE XML JOURNAL

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

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

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

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

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

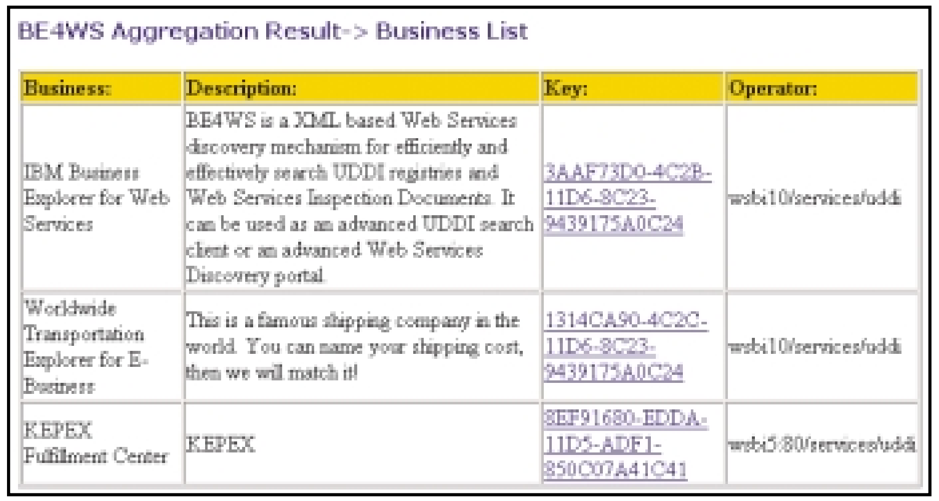
CUSTOMER RELATIONS / JDJ STORE
MANAGER, CUSTOMER RELATIONS
Anthony D. Spitzer tony@sys-con.com
CUSTOMER SERVICE REPRESENTATIVE
Margie Downs margie@sys-con.com
MANAGER, JDJSTORE
Rachel McGouran rachel@sys-con.com

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

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

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

SYS-CON MEDIA



BE4WS Aggregation Result-> Business List

Business:	Description:	Key:	Operator:
IBM Business Explorer for Web Services	BE4WS is a XML based Web Services discovery mechanism for efficiently and effectively search UDDI registries and Web Services Inspection Documents. It can be used as an advanced UDDI search client or an advanced Web Services Discovery portal.	3AAF73D0-4C2B-11D6-8C23-9439175A0C24	wsbi10/services/uddi
Worldwide Transportation Explorer for E-Business	This is a famous shipping company in the world. You can name your shipping cost, then we will match it!	1314CA90-4C2C-11D6-8C23-9439175A0C24	wsbi10/services/uddi
KEPEX Fulfillment Center	KEPEX	8EF91680-EDDA-11D5-ADF1-850C07A41C41	wsbi5:80/services/uddi

FIGURE 4 Aggregated result from BE4WS

Web Services (BE4WS) – since last December. I believe other companies will follow up to develop similar technologies or systems. Currently, BE4WS has been enhanced to support UDDI version 2 and SOAP version 3 (i.e., AXIS). You can get the latest technology release of BE4WS from Web Services Toolkit version 3.1 or later.

What are the major differences between using BE4WS and using a regular UDDI client? A UDDI client like the UDDI4J typically provides only lower-level APIs for searching the UDDI registry. BE4WS provides higher-level APIs to take advantage of UDDI4J or other clients, such as the WS-Inspection search tool, to define a script-based search request, aggregate search results, and explore multiple UDDI registries concurrently. For example, a script-based single USML (UDDI Search Markup Language) request in BE4WS can include multiple queries, different UDDI sources, and an aggregation operator. The returned result is aggregated from different UDDI registries or other sources according to the search criteria defined in the USML script. If you're using UDDI4J alone, you must write your own code to send out multiple search queries to different UDDI registries and then aggregate the data yourself. Obviously, this is a time-consuming process, and an advanced UDDI search engine can better handle such requests.

BE4WS can incorporate not only different Web services discovery technical layers (such as UDDI for Java technology or other similar UDDI client packages), but also different sources (such as standard UDDI registries, WS-Inspection documents, etc.). So BE4WS enables you to interact with XML representations of a UDDI exploring mechanism instead of working directly with UDDI for Java technology and other UDDI clients. With BE4WS, you can use the same programming model regardless of how your Web services discovery clients are implemented. BE4WS could emerge as a standard efficient and effective exploring mechanism and common interface for UDDI registries and other sources.

BE4WS provides two types of APIs for business application developers: a regular Java API and a Web services interface.

Three pubic methods have been exposed in the BE4WS: searchByFi leName, searchByString, and searchByURL. Any e-business application developer can use these APIs to easily find desired business entities, services, and service types (tModels) from different UDDI registries using a single Java or SOAP call. The only input parameter for these calls is a USML request string or a USML document location. The returned result is also a USML response document. Thus, BE4WS can be easily deployed as a central UDDI search portal. All authorized e-business applications can take advantage of this service to explore business and service information across multiple UDDI registries.

Once a BE4WS Web service is published in a private or public UDDI registry, any application can find this useful UDDI search service and invoke it through a WSDL interface. The installed BE4WS Web service works as an advanced UDDI search portal that allows SOAP clients to search for business and service information in multiple UDDI registries. There are two methods in the BE4WS Web service: searchByString and searchByURL. The SOAP service ID is urn:uddisearch-service, and the provider class is com. ibm.UDDI.BusinessExplorer. UDDI Search.

Listing 1 illustrates a USML script. There are two queries in one USML request, and this request will be sent to BE4WS. The first query represents the search requester who wants to find business entities whose names include Explorer from Private UDDI Registry 1 installed on wsbi5. The second query represents the search requester who wants to find business entities whose names start with KEP from Private UDDI Registry 2 installed on wsbi10. The results from these two UDDI registries will be aggregated according to the definition of AggOperator (Aggregation Operator) in the USML script. In this example the two results will be united and sent back to the search requester. The Aggregation Operator can be defined as "OR", "AND", or "Script". The detailed description of the tags in the USML can be found in the BE4WS informayion included in the WSTK 3.1 release.

After the user creates the above simple script, named BE4WS-example.xml, he/she can put this file into the local system or on a Web server. If this USML script is stored on the local system, the user simply needs to type the following command to execute the BE4WS engine:

```
Java services.demos.be4ws.client.Demo1
BE4WS-example.xml
```

Listing 2 shows the search result – a typical business list composed from two UDDI registries.

As mentioned earlier, BE4WS can be deployed on an application server as an advanced Web services search portal for B2B integration or Web application integration. For example, the federated UDDI search request defined in Listing 1 could be used by an enterprise application, such as an intelligent service locator (agent), to look up UDDI registries for finding desired services providers. The aggregated search result (USML response) shown in Listing 2 can be parsed and presented in a JavaServer Page (JSP) created by the intelligent service locator. Figure 4 shows the aggregated result from BE4WS.

This example demonstrates a simple USML script for searching multiple UDDI registries. This method simplifies the developer's programming work and provides a more accurate result aggregated by BE4WS according to the search criteria defined in the USML script.

Conclusion

We know the existing Web services discovery mechanisms are designed for very limited use of UDDI or WS-Inspection documents. The BE4WS-like, script-based Web services discovery mechanism with result aggregation from different sources – a key enabling technology for e-business integration providers – will be the next big thing in the integration of Web services applications.

Listing 1

```
<?xml version="1.0"?>
<!DOCTYPE Search SYSTEM "UDDISearch.dtd">
<Search>
 <ProcessId>9999</ProcessId>
 <Query>
  <Source>Private UDDI Registry 1</Source>
  <SourceURL>http://wsbi10/services/uddi/inquiryAPI</SourceURL>
  <BusinessName>%Explorer</BusinessName >
  <FindBy>Business</FindBy>
 </Query>

 <Query>
  <Source>Private UDDI Registry 2</Source>

<SourceURL>http://wsbi5/services/uddi/servlet/uddi</SourceURL>
  <BusinessName>KEP</BusinessName >
  <FindBy>Business</FindBy>
 </Query>

 <AggOperator>OR</AggOperator>
</Search>
```

Listing 2

```
<?xml version="1.0"?>
<USMLResponse>
 <Business>
  <BusinessName>IBM Business Explorer for Web
     Services</BusinessName>
  <BusinessKey>3AAF73D0-4C2B-11D6-8C23-94
     39175A0C24</BusinessKey>
  <Description>BE4WS is an XML based Web services discovery
     mechanism for efficiently and effectively search UDDI
     registries and Web Services Inspection Documents. It can
     be used as an advanced UDDI search client or an advanced
     Web Services Discovery portal.</Description>

<URL>http://wsbi10/services/uddi/uddiget?businessKey=3AAF73D0-
4C2B-11D6-8C23-9439175A0C24</URL>
  <Operator>wsbi10/services/uddi</Operator>
 </Business>
 <Business>
  <BusinessName>Worldwide Transportation Explorer for E-
     Business</BusinessName>
  <BusinessKey>1314CA90-4C2C-11D6-8C23-
     9439175A0C24</BusinessKey>
  <Description>This is a famous shipping company in the world.
     You can name your shipping cost, then we will match
     it!</Description>

<URL>http://wsbi10/services/uddi/uddiget?businessKey=1314CA90-
4C2C-11D6-8C23-9439175A0C24</URL>
  <Operator>wsbi10/services/uddi</Operator>
 </Business>
 <Business>
  <BusinessName>KEPEX Fulfillment Center</BusinessName>
  <BusinessKey>8EF91680-EDDA-11D5-ADF1-
     850C07A41C41</BusinessKey>
  <Description>KEPEX</Description>
  <URL>http://wsbi5:80uddiget?businessKey=8EF91680-EDDA-11D5-
     ADF1-850C07A41C41</URL>
  <Operator>wsbi5:80/services/uddi</Operator>
 </Business>
</USMLResponse>
```

Download the code at

sys-con.com/webservices

Web Services Management Solutions

IT'S TIME TO START PLANNING

Many companies are in the initial planning and pilot stages with Web services initiatives and may not think they need to consider management solutions right now. They need to think again.

> "Understanding the features and benefits of Web services–management technologies available today will have a significant impact on how Web services are developed and how companies can use Web services tomorrow"

Understanding the features and benefits of Web services–management technologies available today will have a significant impact on how Web services are developed and how companies can use Web services tomorrow. The news may be much better than most think, especially if concerns about security, reliability, connection provisioning, and monitoring are making executives shy about taking the Web services plunge.

Before I talk about different types of management platforms, it may help to take a look at the problems management platforms address. Web services–management solutions exist because Web services are cross-platform and potentially cross-organization in nature. While the stack of Web services standards provides the basis for cross-platform and cross-organization interoperation, companies still need an infrastructure to manage the "distributed system" created by the enterprise.

Further, today's Web services standards are immature, so they have to be augmented with other technologies to ensure security and reliability. Because they're so new, the standards are sure to evolve. Management platforms act as a "shock absorber," buffering companies from the impact of future changes in standards while enabling Web services for business benefit now.

What Web Services Management Platforms Provide

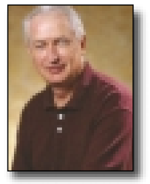
Management platforms can be separated into two categories based on the focus of their management-feature set. One category focuses on managing the Web services themselves, addressing such requirements as version management, performance monitoring, and composite Web services workflow. These solutions abstract Web services from the execution platforms and provide a layer of control that spans multiple underlying technologies.

The second category focuses on the actual "connections" between Web-services providers and requesters. This connection abstraction includes not only the Web service and its transport binding, but also the relevant infrastructure details of the provider and requester endpoints.

Just as any given Web service has its own unique characteristics, each of the multiple connections associated with it has its own characteristics. Potentially, each connection can have different combinations of authentication, encryption, reliability and other protocols as well as service-level definitions plus environmental variables such as firewall and proxy configurations associated with provider and requester endpoints.

These management platforms are often referred to as Web services–networking solutions. These solutions address three critical functions in the implementation and management cycle: connection provisioning, optimized SOAP communications, and monitoring network services.

Connection provisioning is the critical step in a Web-services implementation cycle where technology meets processes and people and often represents the biggest bottleneck of Web services projects. Provisioning requires a level of coordination and agreement between provider and requester regarding the technical details associated with each connection. Yes,



Author Bio

A background steeped in telecommunications, networking, enterprise applications, and e-business solutions led Dave Spicer to the creation of Flamenco Networks, where he serves as CEO and CTO. Flamenco Networks is a provider of Web services-networking solutions.

DAVE.SPICER@FLAMENCONETWORKS.COM

two developers can provision a connection over the phone; however, if a Web service involves more than a handful of requesters, that manual process will never work.

A Web services–networking solution automates provider/requester interactions with a self-service browser-based application, then codifies their agreement and synchronizes their infrastructures to speed implementation of the connection and simplify maintenance.

At runtime, the network acts as an intelligent interceptor of SOAP messages and handles and implements the complex, interrelated aspects of transport protocol, including authentication, signing, encryption, reliability, compression, streaming, and state management, based on the specific connection parameters previously declared during the connection provisioning process. A Web services networking solution provides an "edge cache" where PKI information and the declarative parameters of each Web-services connection are stored, so runtime performance isn't hampered by calls to third-party verification servers.

Finally, the Web services–networking solution provides monitoring and network services. It selectively gathers log data, traps error conditions, and allows customers to view all network connections from a single dashboard. Because the Web services–networking solution monitors both ends of the connection, it's able to correlate a Web-services transaction across platforms and even across enterprises to give providers and requesters a holistic view of statistics from the entire network.

Once companies realize the necessity of a management infrastructure, a common question is, "Can't I just build my own?" Some of the functionality provided by a Web services–networking solution can be developed internally, but building it can potentially quadruple the costs of a Web services project and, perhaps just as important, significantly slow time to market. Given that, most companies opt to utilize a third-party solution.

Characteristics and Considerations

A number of vendors offer Web services–management solutions as either a subscription-based service or as licensed software. Some subscription-based service providers deliver transparent infrastructure, while others act as intermediaries to whom companies outsource their B2B integration tasks. In many cases, companies will require a combination of services- and network-management technologies. Selecting solutions that best meet the needs and objectives of the project depends on many factors, including how the Web services project is deployed and who the users are.

Pick the Low-Hanging Fruit

These days, it's hard to find an IT trade publication that isn't devoting a significant portion of its editorial space to Web services, and many are publishing articles addressing where Web services fit in the overall IT scheme. For a breakthrough project, companies should look to areas where Web services can deliver significant ROI with only a minor investment, making the decision easy – in sales and marketing vernacular, low-hanging fruit (LHF).

Upon examining those companies that have identified such projects, we see some common threads emerging:

- ***No replacement:*** These projects rarely involve replacing existing solutions such as EDI or EAI. Rather, they focus on integration challenges that existing solutions could never solve due to technical or cost-related issues.
- ***Big company drives:*** As was the case with EDI, these Web services projects are often driven by a large company to a community (either internal or external) of parties that have a vested interest in acting on the "invitation" of the big company. Few, if any, of the projects that deliver immediate benefit are based on the "build it, publish it to UDDI, and wait for someone to find it" model.
- ***Expose existing applications to trusted parties:*** Rather than new applications built from scratch with Web services, LHF projects are often based on wrapping existing applications to expose them as Web services. These Web services projects are almost always limited to trusted parties, usually exposed internally first and then rolled out to business partners with whom there is a preexisting relationship.
- ***High scale, low touch point:*** High-scale/low touch-point projects are particularly suited for Web services because they involve a large number of users with relatively few and unsophisticated interfaces. For example, an insurance company uses Web services to provide coverage verification – a simple transaction – to 10,000 healthcare providers, a high number of users.

Historically, organizations have often had to limit electronic workflows to their largest business partners because it wasn't cost effective to include all parties. Now, by implementing these high-scale/low-touch-point Web services projects, they have the opportunity to involve everyone.

Conclusion

In all cases, a Web services–management infrastructure is necessary. Just imagine the chore of manually provisioning connections in a high-scale/low touch-point project involving hundreds or even thousands of users with various IT environments and expertise. The resources required for that function alone would overwhelm even the most sophisticated IT groups.

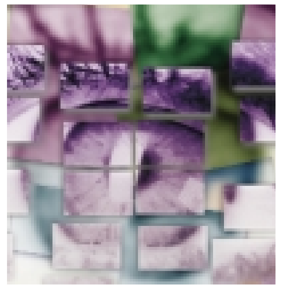
Web services–management platforms give companies the ability to accelerate the connection process, and that speeds the realization of business benefits. Furthermore, they enable enterprises to control and coordinate their Web services when they are in production – these are critical functions because when Web services are in production, they involve multiple infrastructures and organizations. For any company seeking business benefits through the use of Web services, management platforms represent critical infrastructure that should be considered early planning stages.

FOCUS P2P MESSAGING

Written by Milan Milenkovic

The Future of Internet Distributed Computing

Services that are reusable and recursive



The current implementation of the World Wide Web and its supporting Internet infrastructure is predominantly geared toward information retrieval and display in a humanly readable form. Its data formats and protocols are neither intended nor very suitable for automated machine-machine interactions without humans in the loop.

Emerging uses of the Internet – including peer-to-peer and grid computing – provide both a glimpse of and the impetus for evolution of the Internet into a distributed computing platform of unprecedented scale.

In this article I explore the needs and requirements of distributed computing on the Internet and present a case for a building-block approach that amalgamates and extends the foundation provided by standards-based Web services, peer-to-peer, and grid computing. I analyze the requirements of emerging and predicted modes of Internet distributed computing and extrapolate some fundamental underlying principles and mechanisms.

When implemented, the proposed building-block approach will provide an unprecedented level of modularity and standards-based infrastructure support for composition of distributed applications on the Internet that combine the best aspects of Web services, peer-to-peer, and grid computing.

AUTHOR BIO:
Dr. Milan Milenkovic is director of distributed systems architecture in Intel's Corporate Technology Group (R&D). He has authored several textbooks and numerous articles in the area of systems design.
MILAN@INTEL.COM

Web Services

Distributed computing on the Internet, in the form of Web services, is rapidly evolving toward the ability to dynamically link dispersed, heterogeneous platforms into self-organizing structures to provide a plethora of services. These include both end-user services and automated, machine-to-machine, programmatic services. Web services are self-contained, self-describing, loosely coupled software components that can be described, published, discovered, and invoked over the Internet using standard protocols. They can dynamically locate and interact with other Web services on the Internet to provide a complex service without human intervention.

Web services are platform-independent and based on Internet standards – such as XML, SOAP, UDDI, and WSDL. Coupled with the ubiquitous connectivity of the Internet, Web services realize a long-sought vision of component software: distributed, reusable, standardized, open, scalable, and Internet-centric. The distribution of computer resources across the Internet is an acknowledged trend in today's computing industry. XML and Web services are often seen as providing the "plumbing" for that distribution.

Peer-to-Peer Computing

Peer-to-peer computing provides both a design approach and a set of operating assumptions that enables exploitation of the aggregate power of millions of networked personal computers and high-end personal digital assistants. It provides file backup, content distribution, and collaboration without the need for centrally provisioned and managed services. Peer-to-peer solutions are also noted for dealing with practicalities found in vastly heterogeneous collections of personal machines, such as intermittent node availability, non-DNS IP addressability (due to dynamically assigned and translated IP addresses) and bidirectional communication through Network Address Translation (NAT) units and firewalls.

Grid Computing

Grid computing extends conventional distributed computing by large-scale sharing of computational and storage resources among dynamic collections of individuals and institutions. Such settings are characterized by unique scale, security, authentication,resource access, and resource discovery requirements.

The term grid comes from the notion of computing as a utility and an analogy with a power grid as a pool of resources combined to meet variations in load demand without users' awareness of (or interest in) how it is done.

Grid's origins and most of its current applications are in the area of high-performance computing.

Internet Distributed Computing

Internet Distributed Computing (IDC) describes the use of the Internet as a distributed computing platform by combining and unifying common aspects of Web services, peer-to-peer, and grid computing, while preserving their unique problem-solving aspects. Internet standards and connectivity are used to achieve convergence, interoperability, and reuse of a set of common building blocks that meet the listed and emerging requirements of distributed computing on the Internet.

The benefits of distribution and decentralization of control and resources – such as objects, storage, and processing power – are inherent in the new model of IDC. We cover the benefits of functionally rich clients in implementing Web services in both symmetrical (peer-to-peer and grid computing) and asymmetrical (resembling client/server computing) application architectures. While powerful clients offer obvious benefits for individual users, for the purposes of cooperating on a common task they can also dynamically form a powerful union of even global proportions, such as planetary grids.

In addition to the common building blocks, IDC embodies a set of fundamental abstractions and mechanisms.

Resource Virtualization

A service in the world of Web services is essentially a form of virtualization of a software functional component. Services can be advertised and discovered using directories, such as UDDI, and inspection, such as WSIL. Once discovered, an invoking entity can bind to the selected service and communicate with its externally visible functions via protocols such as SOAP. These concepts can be extended to hardware to virtualize resources, such as compute cycles, storage, and devices. Each virtualized component can be abstracted as a Web service that can be advertised, discovered, and bound.

This deceptively simple extension transforms the (software) component model into a distributed component model of immense power. With its application carried to a logical extreme, one can envision a planetary-size pool of composable hardware and software resources connected via the Internet and described and accessible using Web-service protocols and conventions.

Dynamic Configuration and Runtime Binding

Dynamic configuration has several aspects that are useful in different application domains. Its foundation is the ability to perform runtime binding of components, as opposed to, say, design time or link time. Deferred or runtime binding may be implemented through a combination of Web-service discovery and binding mechanisms. Its primary benefit is decoupling of the application design and execution from awareness of the underlying system configuration and physical connectivity. This in turn allows application portability across a wide range of platforms and network configurations. It also allows decoupling of development of users and providers of services, unlike the tight tandem-development with shared logic inherent in client/server applications.

Runtime binding also enables desirable system capabilities such as load balancing (by binding to a least-loaded service from a functionally equivalent group) and improved reliability (by binding to a service that is available at time of invocation).

Finally, runtime binding is a necessary ingredient for ad hoc and self-configuring networks. These are sometimes referred to as zero-configuration networks, which is how they appear to users. This capability facilitates use via automatic peer configurations in systems with high node-fluctuation rates.

Resource Aggregation and Orchestration

The primary purpose of creating a networked, configurable pool of virtual resources is to be able to dynamically aggregate the exact collection needed to perform a task at hand, such as execute a specific application. Runtime resource aggregation to meet application resource requirements implies that applications are designed in a manner that allows them to state their requirements or at least provide hints to the execution system that enable it to arrive at a reasonably efficient estimate.

Resource aggregation is already being used in grid computing and high-performance cluster configurations. It will also play a key role in implementing the vision of pervasive computing with computing resources embedded in the "rich computing" environment.

For this discussion, resource orchestration refers to control and management of an aggregated set of resources so that they can make collective progress towards task completion. It also includes the communication and synchronization necessary for coordination and collating of partial results. Naturally, following completion of a task, resources can be released back to the pool for allocation to other uses.

Security and Authentication

Security and authentication, while not a unique IDC requirement, are necessary for most applications. Resource sharing and aggregation across potentially distinct security domains and levels of trust necessitate protection of both the host and the guest application. Moreover, application execution over a collection of components may require a single system-wide sign-on, as opposed to unwieldy authentication at each individual node.

Future Applications

Future evolution of IDC is expected to include pervasive and proactive computing. The pervasive computing vision postulates an order of magnitude greater scale of Internet "things" – such as sensors, cars, and home appliances. This requires expanded naming and addressing schemes, such as IPv6, and force application designs where intermittent node availability is a default operating assumption, as opposed to a failure mode which is the prevalent design assumption today.

Pervasive computing is often associated with the creation of "smart" environments with embedded computing resources. IDC will enable mobile users to carry a subset of physical computer resources that they need to augment their computational, storage, and UI capabilities when required by dynamically aggregating resources found in the environment. Pervasive and proactive computing include the notion of real-world awareness via sensors. Proactive computing also adds the ability to initiate certain actions automatically and autonomously based on a specific context, such as: ambient conditions, user's intent, preferences, or location. Context may be deduced

by the system, explicitly designated by the user, or derived through a combination of these. Context awareness can be useful in any system, but it becomes essential as the number of computing devices operating on a user's behalf grows by orders of magnitude in the world of Internet of things.

While not discussed here, emerging hybrid wired/wireless LAN connectivity continues to have significant influence on IDC design. There is evidence that Web services will be the predominant set of standards governing e-business and grid computing. By adopting and extending these standards to the mobile space early, we hope to accelerate the adoption of mobile computing and associated technologies by providing a natural bridge to an information ecosystem capable of engendering advanced (e.g., proactive) services and virtualized resources. Other factors motivating IDC design include the desire for efficient use of computing resources and hiding from users the complexity inherent in managing heterogeneous distributed systems. These are evidenced by emergence of utility computing, autonomic computing, and massively parallel applications such as peer-to-peer, grid-like formations that aggregate resources from millions of personal computers.

IDC is poised to accelerate distributed computing on the Internet by providing an environment to aggregate, discover, and on-demand, dynamically assemble computing structures (services and resources) for the task at hand. Its underlying principles are reusable and recursive, i.e., they can be "reincarnated" at different scales in sensor networks, home-area networks (UPnP), enterprise networks, and wide-area networks. Indeed, IDC will provide a foundation for pervasive computing from small-scale personal area networks to virtual, planetary-scale grids.

References

- *Web Services Architecture Requirements, Editor's Draft 26 June 2002:* www.w3.org /2002/ws/arch/2/06/wd-wsa-reqs-2002 0605.html.
- *Web Services Description Language (WS DL) Version 1.2, W3C Working Draft 9 July 2002*: www.w3.org/TR/2002/WD-wsdl12-20020709.
- *Web Services Description Language (WS DL) Version 1.2: Bindings, W3C Working Draft 9 July 2002:* www.w3.org/TR/2002/ WD-wsdl12-bindings-20020709.
- *SOAP Version 1.2 Part 0: Primer, W3C Working Draft 26 June 2002*: www.w3.org/ TR/2002/WD-soap12-part0-20020626.
- Neel, Dan. (2002) "The Utility Computing Promise," *InfoWorld*: www.infoworld.com/articles/fe/xml/02/04/15/020415feutility .xml.
- IBM Research. "Autonomic Computing: IBM's Perspective on the State of Information Technology," www.research.ibm.com/autonomic/manifesto.
- Foster, I. et al. (2002) "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," June. www.gridforum.org/ogsi-wg/drafts/ogs a_draft2.9_2002-06-22.pdf.
- *Global Grid Forum*, www.gridforum.org.
- D. Barkai (2001). "An Introduction to Peer-to-Peer Computing," *Intel Developer Update Magazine*. February. www.intel .com/update/departments/initech/it02012.pdf.
- *The 1st International Workshop on Peer-to-Peer Systems* (IPTPS '02), www.cs.rice.e du/Conferences/IPTPS02.
- Satyanarayanan, M., "Pervasive Computing: Vision and Challenges," *IEEE Personal Communications Magazine* (2001), pp. 10-17, August. www.comsoc.org/pubs/pcm/index.html.
- D. Tennenhouse (2000). "Proactive Computing," Communications of the ACM Vol. 43. No. 5. www.acm.org/pubs/articles/journals/cacm/2000-43-5/p43-tennen house/p43-tennenhouse.pdf.

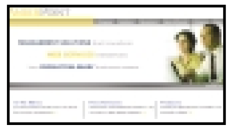
webMethods Announces Support for BPEL4WS Specification

(Fairfax, VA) – webMethods, Inc., provider of integration software, has announced support for the Business Process Execution Language for Web Services (BPEL4WS) specification, created jointly by Microsoft, IBM, and BEA Systems). BPEL4WS is an XML-based language that describes how Web service–based business processes interact within and between enterprises.

BPEL4WS enables companies to define business processes and ensure interoperability across the enterprise and with business partners in a Web services environment. The specification replaces IBM's WSFL and Microsoft's XLANG efforts by combining and extending the functions of these foundation technologies. www.webMethods.com.

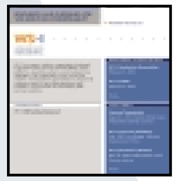
AmberPoint Joins Web Services Interoperability Organization

(Oakland, CA) – AmberPoint, Inc., has joined the Web Services Interoperability Organization (WS-I), demonstrating the company's leadership role in Web services management. AmberPoint will contribute its experience in distributed computing and



management frameworks, and assist the WS-I community in developing tools, resources, and other guidance to support Web services implementations.

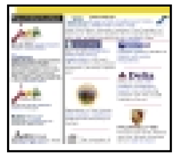
The WS-I is committed to promoting interoperability across Web services platforms, applications, and programming languages. AmberPoint joins more than 125 other companies that have become



members of WS-I since its official launch in February 2002. As a member, AmberPoint gains access to tools, materials, and conformance tests that will ensure its products as well as its customers' Web services are compatible and interoperable. www.amberpoint.com, www.ws-i.org.

Jacada Adds Full Web Services Support to Jacada Integrator

(Atlanta) – Jacada Ltd. has announced general availability of Jacada Integrator 3.6, which adds full Web services support to Jacada's legacy integration offering. Without altering the existing application, the new functionality exposes existing business logic and data locked up in legacy systems as open, reusable Web services.



With this new release, Jacada Integrator offers a risk-free approach to testing and implementing Web services. Jacada Integrator provides all the necessary components for creating and using Web services including the automatic generation WSDL files and a SOAP listener for invoking the new Web service. www.jacada.com.

Vultus Launches First Browser Application Platform

(Orem, Utah) – Vultus, Inc., has announced the availability of its WebFace Solution Suite. This product suite, consisting of the WebFace Browser Application Platform and WebFace Studio, is designed to accelerate the delivery of enterprise applications over the Web.

The WebFace Solution Suite delivers a Web-based presentation layer that mirrors the traditional desktop computing environment and removes many of the

obstacles associated with static HTML Web pages. Utilizing widely accepted industry standards, XML, JavaScript, and HTML, the suite renders full-featured application windows inside a Web browser. WebFace does not rely on third-party plug-ins and does not reside locally on the end-user's machine. www.vultus.com.

BEA Expands Web Services with JAX-RPC Standard

(San Jose, CA) – BEA Systems, Inc., a leading application infrastructure software company, has announced support for a certified implementation of the Java API for XML-based Remote

Procedure Call, making BEA the first major application server provider to adopt the new standard for Web services interoperability. Their support for JAX-RPC in BEA WebLogic Server will help developers more rapidly achieve Web services interoperability across all industry standards and protocols.

JAX-RPC helps enable quick interoperability between Web services, regardless of the underlying platform, meaning that Web services can communicate with one another even if they reside on different platforms or were created in different programming languages. www.bea.com.

Recursion Software Launches C# Rapid DevKit

(Dallas) – Recursion Software, Inc., a provider of software development solutions, has announced availability of its C# Rapid DevKit. The C# Rapid DevKit includes high-performance enterprise collections and more than 50 advanced data processing algorithms that reduce .NET development time.

The C# Rapid DevKit extends the .NET Framework and is compatible with the ICollection and IEnumerator classes. It also includes eleven highly optimized data structures that meet the needs of the most demanding programmers. With its time-saving algorithms, C# Rapid DevKit provides developers with high-performance, full-featured, and easy-to-use extensions for the basic .NET framework. www.recursionsw.com.

Cysive Releases Cysive Cymbio 2.3 Interaction Server

(Reston, VA) – Cysive, Inc., has announced the general availability of the Cysive Cymbio 2.3 Interaction Server, an integrated, multichannel Web services platform that allows enterprises to communicate and interact with customers, partners, and employees over multiple communications channels.

An evaluation version of Cysive Cymbio 2.3 is available for download at http://cymbio.cysive.com. This version allows developers to download and install Cysive Cymbio and develop a sample multichannel application in less than four hours. www.cysive.com.

WSJ ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
Actional	www.actional.com/whitepapers/wsjblues	800-808-2271	4
Altaworks	www.altaworks.com	603 598-2582	21
Altova	www.xmlspy.com		15
BEA	www.bea.com		68
BEA WebLogic Developer's Journal	www.weblogicdevelopersjournal.com	888-303-5282	56
engindata Research	www.engindata.com	201-802-3082	57
Gartner	www.gartner.com/us/eai	800-778-1997	29
IBM	www.ibm.com/websphere/integrate		2
JDJ Store	www.jdjstore.com	800-303-JAVA	33, 45
Macromedia	www.macromedia.com		19
PolarLake	www.polarlake.com		25
.NET Developer's Journal	www.sys-con.com/dotnet	888-303-5282	51
Sitraka	www.sitraka.com/jclass/ws		9, 67
Sonic Software	www.sonicsoftware.com/websj		6
SpiritSoft	www.spiritsoft.com/climber		11
SYS-CON Events	www.sys-con.com	201-802-3069	41
SYS-CON Media	www.sys-con.com	888-303-5282	49, 61, 65
SYS-CON Reprints	www.sys-con.com	201-302-3026	54
Web Services Journal	www.sys-con.com	888-303-5282	37
WebSphere Developer's Journal	www.sys-con.com	888-303-5282	55

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.

GRID COMPUTING: Electrifying Web Services

Combining Web services and grid computing will virtualize networked resources

Written by Dirk Hamstra

Grid computing makes it possible to dynamically share and coordinate dispersed, heterogeneous computing resources. Flexibility and ubiquity are essential characteristics of Web services technologies such as WSDL (Web Services Description Language), SOAP (Simple Object Access Protocol), and UDDI (Universal Description, Discovery, and Integration).

The Open Grid Services Architecture (OGSA) combines technologies to unlock and exploit grid-attached resources. OGSA defines mechanisms to create, manage, and exchange information between Grid Services, a special type of Web service. The architecture uses WSDL extensively to describe the structure and behavior of a service. Service descriptions are located and discovered using Web Services Inspection Language (WSIL). By combining elements from grid computing and Web services technologies, OGSA establishes an extensible and interoperable design and development framework for Grid Services that includes details for service definition, discovery, and life-cycle management.

AUTHOR BIO:
Dirk Hamstra has over 15 years of experience in senior-level software development and sales. His area of expertise is application integration and middleware. He has worked at IBM, Vitria Technology, IONA Technologies, and as an independent consultant.
DIRK-HAMSTRA@ATTBI.COM

Drawing the Grid

Information systems are no longer just defined by what they can process, but also where they can connect to. For example, the growing demands for computing power in simulation and engineering design projects, is increasingly satisfied by "on-demand" sharing of CPU cycles and disk space across distributed networks. The ultimate goal for these interconnected networks, or grids, is to make IT-power as commonplace and omnipresent as electricity.

Grid computing can be described as the coordinated, transparent, and secure sharing of IT resources across geographically distributed sites based on accepted computing standards. The grid endpoints include both hardware and software components such as data, files, storage space, and application functionality (see Figure 1). Adding resources to a grid can be difficult today because of the lack of standards and the implementation method. Enabling, for example, monolithic applications to run in a distributed environment, means overcoming issues such as network access performance and interface granularity.

The concept of a grid can be viewed as an abstraction layer or facade, spanning distributed computing resources and enabling virtual collaboration between people and IT processes. Examples of grid implementations are found in computational science, large-scale simulation, data mining, network-enabled solvers and collaborative engineering projects. Lower total cost of computing, access to a broader range of resources, improved flexibility and reliability are the main benefits of using a grids infrastructure.

Current grid projects, such as NASA's Information Power Grid, The GriPhyn, and NeesGrid, are mostly created and operated by universities and government research institutions. One of the earliest and most popular public examples of grid computing is SETI@home. This project uses the processing power of more than 500,000 personal computers from around the world in its quest to search for life in the galaxy.

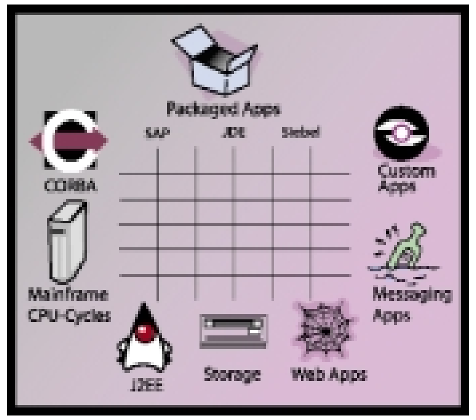


FIGURE 1 Grid endpoints

Grid computing also reflects changing ideas about the nature of software, who owns it, and where it resides. It implements a service-based software model in which computing services are configured to meet a specific set of requirements at a point in time, executed, and discarded, approaching the vision of instant computing.

Broad adoption of grid computing depends on solving technical and economic concerns, including end-to-end security and usage metering. These issues will initially limit the reach of grids to the boundaries of an organization. Over time, partner grids will extend enterprise grids to include external organizations in the value chain. Service grids represent the final phase, in which grids are broadly accepted, making computing resources a utility similar to water, electricity and gas.

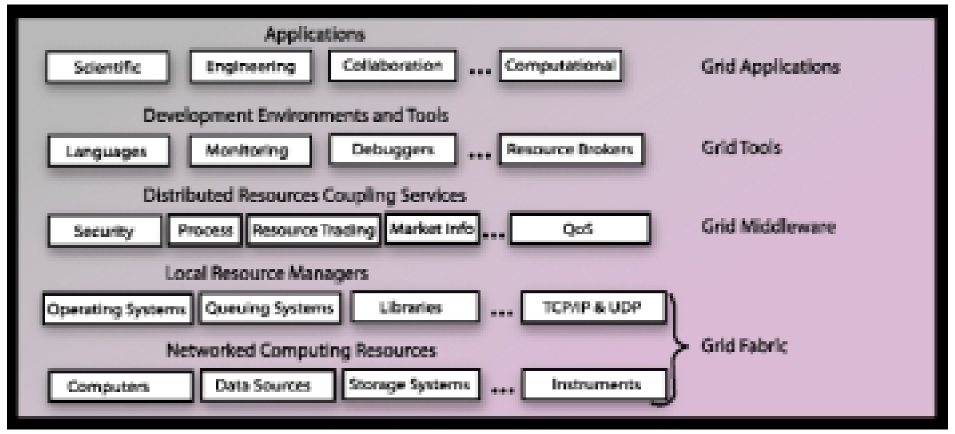
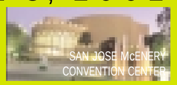


FIGURE 2 Grid Components

Software from commercial companies like IBM, Sun Microsystems, Avaki, and Microsoft; open source initiatives like Globus; and proprietary implementations are used as the building blocks to develop and deploy grid applications. The grid infrastructure is a mix of hardware and software that can be categorized into four groups: fabric, middleware, tools, and applications. The fabric consists of hardware and system software such as operating systems, device drivers, and communication stacks. Grid applications use grid tools and middleware components to interact with the fabric (see Figure 2).

Grids are complex systems as a result of scope and scale factors. Typically, they have large numbers of nodes, service providers, and consumers using different operating systems and applications on a mix of hardware. Usage patterns are unknown, while consistency in the Quality of



Service (QoS) of the response time, for example, is expected. Finally, service providers and consumers can be geographically dispersed and have different objectives regarding service availability and capability. The OGSA defines a framework to develop applications suitable to operate in these large-scale heterogeneous distributed networks.

Grids, Services, and OGSA

From a software-design perspective, a grid can be viewed as a collection of self-contained computing services that can be described, published, located, and invoked over any type of network. Services are location-independent and dynamic, span multiple computing architectures, and reach across administrative domains. The design principles of OGSA reflect a combination of elements from Web services and grid computing:

- ***Resource virtualization:*** Each grid component is considered a service.
- ***Standard interface definition mechanisms:*** Enabling multiple protocol bindings and transparency between local or remote services.
- ***Standard foundation services:*** Defining service semantics, reliability and security models, and core functions such as life-cycle management, or discovery.
- ***Implementation independence:*** Support for multiple development languages and hosting environments including Java, COBOL, C, J2EE, CICS, and .NET.

Since a grid consists of both existing and new hardware and software, multiple variations of communication protocols, security schemes, and transaction management systems exist and cooperate at the same time. Today's grids tend to be a patchwork of protocols and noninteroperable "standards" and difficult-to-reuse "implementations." The OGSA uses Web services technologies like WSDL, SOAP, and WSIL to abstract platform and implementation differences, giving transparent access to grid services.

OGSA is an extension and a refinement of a Web services architecture. It describes a system that normally consists of a few persistent and potentially many transient services. The architecture defines basic service behavior including fundamental semantics, life-cycle management, and discovery. Also included is a framework for fault resilience, security, and transaction management.

The services specified in OGSA can be persistent and time limited, supporting delegated authentication credentials using heterogeneous communications protocols. These characteristics are significantly different from the common Web services model in which services are persistent, allowing anonymous communication over HTTP using SOAP. Transient, short-lived services, dynamically created and destroyed, are an essential component of a grid. Examples include: interfaces to the states of distributed activities, data analysis, and streaming video. Much of the work in OGSA deals with the implications of transient service instances on management, naming and discovery.

TO REGISTER:

ONLINE

WWW.SYS-CON.COM

FAX

201-782-9651

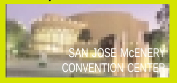
MAIL

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

CALL

201-802-3069

OCTOBER 1-3, 2002
SAN JOSE McENERY
CONVENTION CENTER,
SAN JOSE, CA



YOUR INFORMATION (Please Print) ☐ Mr. ☐ Ms.

First Name________________ Last Name __________________
Title____________________Company __________________
Street ______________________________________
Mail Stop ____________________________________
City ____
State ________Zip ______________Country _____________
Phone ______________________Fax __________________
E-mail ______________________________________

$75 Value

Complete and return this form to receive FREE admission to the Expo and all FREE Show Features.

After September 20, 2002, bring this pass onsite for FREE admission.

PLEASE PRINT CLEARLY
(PHOTOCOPY FOR ADDITIONAL REGISTRANTS).

A. Your Job Title
❑ CTO, CIO, VP, Chief Architect
❑ Software Development Director/Manager/Evangelist
❑ IT Director/Manager
❑ Project Manager/Project Leader/Group Leader
❑ Software Architect/Systems Analyst
❑ Application Programmer/Evangelist
❑ Database Administrator/Programmer
❑ Software Developer/Systems Integrator/Consultant
❑ Web Programmers
❑ CEO/COO/President/Chairman/Owner/Partner
❑ VP/Director/Manager Marketing, Sales
❑ VP/Director/Manager of Product Development
❑ General Division Manager/Department Manager
❑ Other (please specify)

B. Business/Industry
❑ Computer Software
❑ Computer Hardware and Electronics
❑ Computer Networking & Telecommunications
❑ Internet/Web/E-commerce
❑ Consulting & Systems Integrator
❑ Financial Services
❑ Manufacturing
❑ Wholesale/Retail/Distribution
❑ Transportation
❑ Travel/Hospitality
❑ Government/Military/Aerospace
❑ Health Care/Medical
❑ Insurance/Legal
❑ Education
❑ Utilities
❑ Architecture/Construction/Real Estate
❑ Agriculture
❑ Nonprofit/Religious
❑ Other (please specify)

C. Total Number of Employees at Your Location and Entire Organization (check all that apply):

	Location	Company
10,000 or more	01 ❑	01 ❑
5,000 - 9,999	02 ❑	02 ❑
1,000 - 4,999	03 ❑	03 ❑
500 - 999	04 ❑	04 ❑
100-499	05 ❑	05 ❑
100 or less	06 ❑	06 ❑

D. Please indicate the value of computer and communications products and services that you recommend, buy, specify or approve over the course of one year:
❑ $10 million or more
❑ $1 million - $9.9 million
❑ $500,000 - $999,999
❑ $100,000 - $499,999
❑ $10,000 - $99,999
❑ Less than $10,000
❑ Don't know

E. What is your company's gross annual revenue?
❑ $10 billion or more
❑ $1 billion - $9.9 billion
❑ $100 million - $999 million
❑ $10 million - $99.9 million
❑ $1 million - $9.9 million
❑ Less than $1 million
❑ Don't know

F. Do you recommend, specify, evaluate, approve or purchase wireless products or services for your organization?
01 ❑ Yes 02 ❑ No

G. Which of the following products, services, and/or technologies do you currently approve, specify or recommend the purchase of?
❑ Application Servers
❑ Web Servers
❑ Server Side Hardware
❑ Client Side Hardware
❑ Wireless Device Hardware
❑ Databases
❑ Java IDEs
❑ Class Libraries
❑ Software Testing Tools
❑ Web Testing Tools
❑ Modeling Tools
❑ Team Development Tools
❑ Installation Tools
❑ Frameworks
❑ Database Access Tools/JDBC Devices
❑ Application Integration Tools
❑ Enterprise Development Tool Suites
❑ Messaging Tools
❑ Reporting Tools
❑ Debugging Tools
❑ Virtual Machines
❑ Wireless Development Tools
❑ XML Tools
❑ Web Services Development Toolkits
❑ Professional Training Services
❑ Other [Please Specify]

SYS-CON EVENTS, INC, AND SYS-CON MEDIA MAKE NO GUARANTEES REGARDING CONTENT, SPEAKERS OR ATTENDANCE. THE OPINIONS OF SPEAKERS, EXHIBITORS AND SPONSORS DO NOT REFLECT THE OPINION OF SYS-CON EVENTS AND SYS-CON MEDIA AND NO ENDORSEMENT OF SPEAKERS, EXHIBITING COMPANIES PRODUCTS, OR SPONSORS IS IMPLIED.

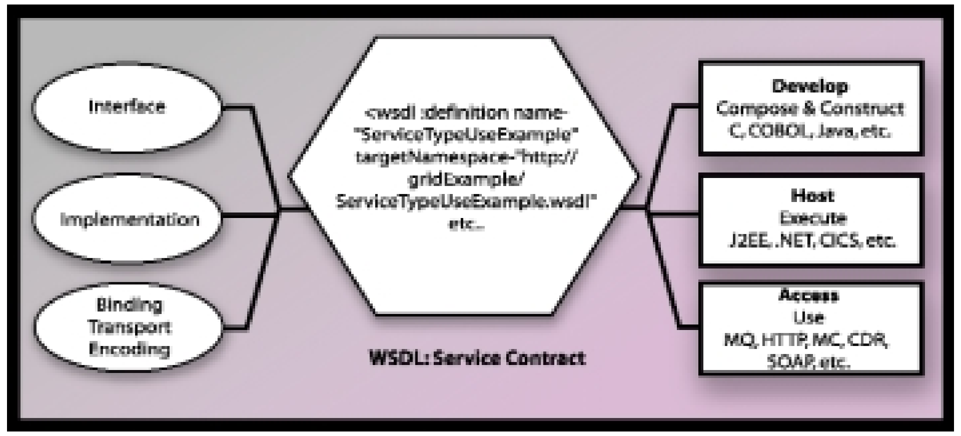


FIGURE 3 **WSDL Service Contract**

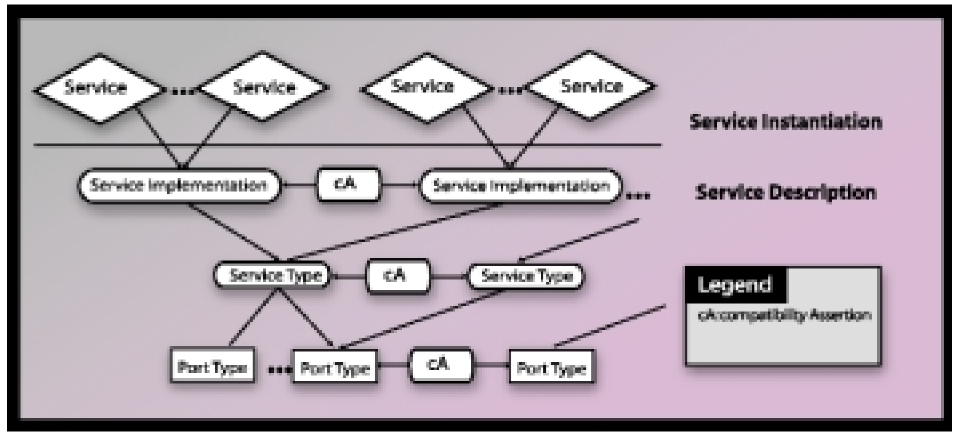


FIGURE 4 **Relationship of WSDL extensions**

Fundamental to OGSA are WSDL and interfaces for dynamic discovery and life-cycle management for a specific type of Web service – a Grid Service. WSDL conventions and extensions are used to describe and structure services, while core service activities are expressed using WSDL interfaces and behaviors. Every Grid Service instance has a unique and immutable name called the Grid Service Handle. Lifetime management is provided by mandatory support for the operations Destroy and SetTerminationTime.

WSDL in OGSA

WSDL enables the creation of "tightly coupled/loosely coupled" systems, separating interface, binding, and implementation. The binding component is further divided in transport, the mechanism used to flow bits from A to B, and encoding, how information is communicated from one system to another. This allows for the separation of concerns for building, hosting and accessing services. A service provider can be developed in C++, hosted on zOS, accessed by a Java application running on Solaris, using SOAP over MQ-Series. WSDL defines the service functionality and an access path or on-ramp to invoke it (see Figure 3).

OGSA introduces the following extensions to WSDL:

- ***serviceType:*** Used to describe aggregated families of services defined by a collection of ports of specified types
- ***serviceImplementation:*** Represents the implementation of the actual code
- ***instanceOf:*** Maps the service description of the actual instance
- ***compatibilityAssertion:*** Associates equivalent interface elements and implementations

A basic WSDL instance document only describes that a service implements a port with the given interface; there is no information about what the service does with that port. The OGSA extensions allow the naming of families of services that have identical semantics and to assert that a particular service implements these semantics. The role of the compatibilityAssertion is to declare that two elements with the same underlying type are indeed compatible even if they have different names. As a result, the clients of the service understand its behavior. Figure 4 illustrates how the WSDL extensions relate to each other.

GridService

A Grid Service is a WSDL-defined service that follows a set of conventions relating to its interface definitions and behaviors. The specification details how clients create, discover, and interact with a Grid Service. Note that OGSA doesn't address the actual creation, management, and destruction of the server component in a particular hosting environment. This means that Grid Services may not be portable across hosting environments but any OGSA-compliant client can invoke them. Three characteristics separate a Grid Service from a regular Web service:

1. A Grid Service is an instance of a service implementation of a service type, a collection of specific interfaces.
2. The instance implements a Grid Services Handle. This is a Uniform Resource Indicator (URI) for the service instance and is bound to a Grid Service Reference (GSR). The GSR is similar, for example, to the Interoperable Object Reference (IOR) in CORBA and contains the necessary information to bind and use the service. The HandleMap interface is used to map between the GSH and GSR. Introducing the GSR in the specification allows the separation of the service name from the service implementation, facilitating service evolution such as "non-stop" service upgrades.
3. The instance also implements a port called GridService supporting three operations:
   - ***FindServiceData:*** Allows a client to discover information about the service's state, execution environment, and additional semantic details not available in the GSR. It is a form of reflection used by the service consumer to learn more about the service.
   - ***Destroy:*** An operation to explicitly destroy an instance
   - ***SetTerminationTime:*** A method to extend the lifetime of a service

In addition to the required GridService port, OGSA also defines additional, optional

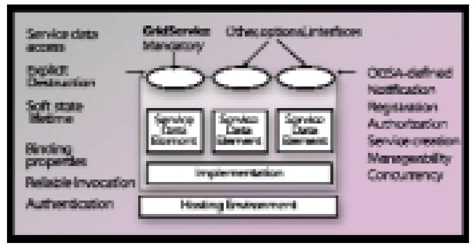


FIGURE 5 | OGSA Ports

service ports. These interfaces define properties commonly required by distributed systems, such as messaging, discovery and registration, instance creation, and full lifecycle management.

The NotificationSource and NotificationSink ports handle messaging. This is a simple publish-subscribe system similar to Java's JMS, using XML messages. Existing messaging systems such as TIBCO or MQ Series can also be used to implement the service. Error notification, application monitoring, and dynamic discovery of services are areas where the Notification ports are useful (see Figure 5).

The Registry port is used to bind the metadata of Grid Service instances, identified by their GSH, to a registry. The registry is a particular type of Grid Service that maintains a set of Grid Service Handles and policies associated with the collection. Extracting information from the registry service is done using the FindServiceData method on the GridService port. This method always supports the XPath query language and returns a WS-Inspection document containing the GSHs of a set of Grid Services. Unfortunately, there is no port type defined to add or remove bindings to and from a service. As a workaround, the notification system can be used.

Instances of a Grid Service can be created via a factory or manually. Similarly, they can be destroyed via soft state termination or explicitly. The factory interface operation CreateService can create transient application service and supports reliable service instantiation (once-and-only-once). Interaction with the factory requires creation-lifetime information and can be extended to include an XML document describing service-specific creation parameters. The interface returns a GSH that uniquely identifies the instance over time. The soft-state lifetime management support avoids clients having to tear down services and prevents resource "leaks" in hosting environments.

Implementations hurdles and future

From a technical perspective, OGSA is critical to streamlining and accelerating the creation and deployment of grid applications. The architecture unifies existing grid development approaches and is extensible from a technical perspective to incorporate future developments. However, to expand the reach of grid applications beyond the level of a single enterprise the OGSA needs to more thoroughly address issues concerning:

- Use of WSDL extensions
- Definition and use of service ports
- Heterogeneous, end-to-end, security
- Grid Service manageability

The extensibility of WSDL has implications for interoperability with existing WSDL clients and servers. Full interoperability and accommodation of non-OGSA WSDL clients by grid servers will accelerate the adoption of the extensibility elements. This is important since the handling of extensions is spotty at best in most current toolkits.

Discovery and lifecycle management deserve a separate interface. In the current OGSA specification, the services are combined in GridService. These services perform distinct roles and are used in different ways. For example, discovery of grid services can be open to anyone, while instantiation and destroy operations require a secure, authenticated connection.

Addressing concerns around authentication, authorization, and trust relations is essential for grid implementations across organizational boundaries. The OGSA should include provisions for "pluggable" security mechanisms that can be discovered by the client using a service description. Service providers, in turn, can plug in a security architecture that fits their infrastructure for Grid Services.

Currently, the OGSA does not indicate how resources are exposed or, in an operational setting, how large sets of Grid Service instances can be monitored and managed. Grid Services are commonly subject to service level agreements that define specific availability, performance, and reliability characteristics. Without agreement on how to instrument and manage, for example, serviceType and portTypes, it will be next to impossible to manage heterogeneous grid applications.

Conclusion

OGSA brings utility-based computing a step closer, and allows the development and deployment of standards-based Grid Services today. The combination of Web services and grid computing virtualizes networked resources using common computing standards, making them accessible to a larger audience. To extend the reach of the architecture and maximize its potential benefits, additional services, such as security and management, need to become an integral part of the specification.

References

- *OGSA*: www.globus.org/ogsa
- *NASA Information Power Grid*: www.ipg.nasa.gov/
- *The GriPhyn*: www.griphyn.org
- *NeesGrid*: www.neesgrid.org
- *SETI*: www.seti-inst.edu
- *IBM*: www-1.ibm.com/grid
- *Sun Microsystems*: wwws.sun.com/software/gridware
- *Avaki*: www.avaki.com
- *Microsoft*: www.microsoft.com
- *Globus*: www.globus.org

Service-Oriented Architectures:

Pushing distributed computing forward

Dave Olander

Dave Olander is VP of engineering at Improv Technologies, a leading developer of enterprise infrastructure software products. Dave has 20 years of experience at HP, Novell, and AT&T Bell Laboratories. His team has just released the latest version of Improv's Cirquet Solution Suite, a life cycle platform for Web services–based distributed applications.

DAVEOLANDER@IMPROV-TECH.COM

The emergence of service-oriented architectures (SOAs) is an exciting development, providing a springboard for the advancement of flexible, dynamic distributed computing solutions. For those not familiar with SOAs, you can think of them as loosely coupled pieces of applications that are published, consumed, and combined with other applications over a network. These are in contrast to existing, tightly coupled distributed computing platforms (such as CORBA, DCOM, or RMI) in that they offer dynamic discovery and binding of application functionality through service descriptions that provide the details on what a given piece of an application does and how its services can be accessed. Sounds a lot like Web services, doesn't it?

Well, Web services represents a form of SOA that is currently making its mark on the IT industry. It is a distributed computing technology that enables flexible, dynamic integration of applications across a network. I'd like to explore what Web services represents to distributed computing, and look at what the broader category of SOAs might offer.

Let's first review the key benefits of distributed computing. In a distributed environment you can move portions of an application under the control of the group responsible for its associated resources. An obvious example would be an inter-enterprise application (e.g., ERP) that spans the control domains of multiple companies. You might also wish to leverage distributed computing for load balancing or failover of your applications. Additionally, in a mobile application you might want to move portions of your application closer to the user, where they will run efficiently. Finally, you may wish to enable peer-to-peer sharing of information or computing resources. In short, distributed computing architectures represent the varied relationships among users and computing resources.

Existing distributed computing platforms have realized many of these benefits. However, the platforms that came before service-oriented architectures tended to be monolithic (despite their object-oriented approach), inflexible, and complex. Developing applications on these systems required expertise in complex programming models, and these systems did not lend themselves to interoperability with other platforms. Often, these platforms restricted the distribution paradigms that could be supported (e.g., lack of peer-to-peer support).

So, isn't that what the buzz around Web services is all about? Freed from the shackles of tightly coupled distributed computing platforms, IT organizations can harness the dynamic capabilities of Web services to achieve their aims, supporting *n*-tier and P2P architectures. Well, yes, sort of. In the realm of application integration, this is certainly true, but distributed computing is about so much more than application integration.

The full potential of SOAs is to more flexibly enable distributed computing at the subapplication, or component, level. Web services that you see in the near future will still suffer from many of the same limitations as tightly coupled platforms because they will continue to be built on these platforms. J2EE and .NET-based Web services will present dynamic service interfaces to monolithic, statically deployed applications. And that's fine at the application integration level.

But I expect to see SOA-based distributed component systems that offer the advantages of a service-based architecture at a finer granularity within the applications. Here's what I think this will look like. First, we will be able to dynamically deploy portions of applications on the network. Imagine having an application component follow a user around a network, from mobile device to PC to wherever. You'll instantiate components when they're needed, where they're needed.

Next, we'll have a location-sensitive communication infrastructure with the intelligence to determine the most efficient means of communication, whether that is direct access to a component service interface on the same system, or an efficient message transport for communication within an enterprise, or SOAP communication for inter-enterprise communication.

You'll be able to update components on the fly to support true, incremental application upgrades. Unlike today's monolithic object-oriented designs, where components cannot be readily replaced without application disruption, SOAs enable dynamic upgrades of portions of an application.

An SOA-based approach will enable the flexibility of both P2P and *n*-tier application architectures to support the natural use pattern of the distributed application.

Web services is a distributed computing technology that is ideal for interconnecting applications across disparate platforms, but it only scratches the surface of the potential for leveraging SOAs for enabling component-level distributed computing. While we should embrace Web services for what they're capable of achieving, we should be pushing the advance of SOAs to take us to the next step in the evolution of distributed computing.

Sitraka

www.sitraka.com/jclass/ws

BEA

www.bea.com