LOOKING AT WIRELESS & WEB SERVICES

January 2002 Volume1 Issue 4

WebServices JOURNAL

.NET J2EE XML

WSJ2.COM

web services EDGE conference&expo

2002WORLDTOUR

BOSTON, MA	JANUARY 29
WASHINGTON, DC	FEBRUARY 26
NEW YORK, NY	MARCH 19
SAN FRANCISCO, CA	APRIL 23

Page 54

PROGRAMMING

Web Services with Scripting Languages

Easy steps, Elegant Methods

Page 6

Wireless .NET page 38

RETAILERS PLEASE DISPLAY
UNTIL FEBRUARY 28, 2002
$6.99US $7.99CAN
0 71486 03420 9 01

SYS-CON MEDIA

Sun

Microsystems

www.sun.com/sunoneinfo

Sun

Microsystems

www.sun.com/sunoneinfo

IONA

www.iona.com

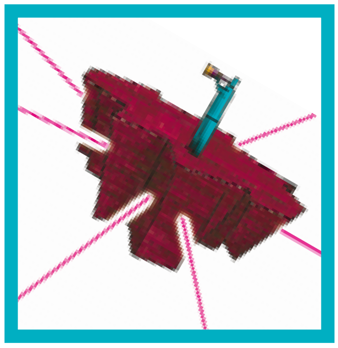
It's a Wireless, Wireless World

written by
Sean Rhody

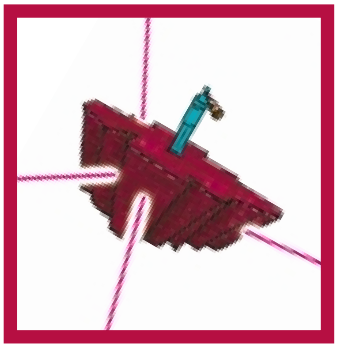
Author Bio:
Sean Rhody is the editor-in-chief of **Web Services Journal**. *He is a respected industry expert and a consultant with a leading Internet service company.*

SEAN@SYS-CON.COM

Wireless, like beauty, is in the eye of the beholder. Mention wireless, and you can step back and watch the conversation spin for hours around differing definitions and approaches. In some minds, wireless is all about cell phones, and consequently is a completely consumer-oriented market. To others, wireless includes a much larger host of technologies, including things like wireless networks, PDAs, cell phones, and other embedded or proprietary devices, things like the pad UPS hands you to sign for your delivery. And to some it's a question of consumer versus industrial applications.



Given this diversity within the wireless market itself, it's not surprising that the concept of Web services on wireless is one that can be interpreted in a number of ways. This is especially true in light of the fact that we can't really get one firm definition of what Web services is. If you doubt that, take a look at our Web site and listen to the two panel discussions from Web Services Edge East and West.

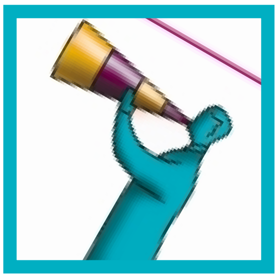


So what does wireless mean to Web services, or what does Web services mean to wireless? That also depends largely on the perception of both categories. To Web services, wireless represents a great opportunity for mass marketing of services and the greatest probability of driving widespread consumer adoption of Web services by wrapping them in applications that support PDAs, cell phones, Blackberrys, occasionally connected PCs, and other devices that communicate without wires.

Wireless also represents a cutting edge area where the deployment of Web services can showcase both the ease of deployment and the viability of platform-independent services. Wireless may be the one particular application of Web services that drives widespread use and adoption at the individual user level.

And to some extent that's important to all of us who see Web services as the hope of people for the next "HTML." While Web services are more complex, they need not baffle the end user, and the ability to craft interesting pieces of code, just like the ability to craft interesting pages, may drive a true revolution in where Web services are positioned. Rick Ross, founder of the Java Lobby ,said it best when he said "Web services should be fun. They should be cool. They should be about being able to do cool things on my PC and sharing them with the world." While I see many obstacles to such a plan, I can't help but applaud that ideal – making Web services so easy my parents could build them.

But what's in it for wireless? What makes Web services interesting to a wireless user? It's that portal into the wide world of applications. The ability to make wireless devices the end user interface to a world of Web applications, all with a simplified interface that can make doing business with a PDA or a lightweight laptop viable, by reducing the data that has to go over the Web. Stripping off the browser, and making whatever application can call a Web service the client, freeing the service developer from worrying about the presentation, and freeing the presentation developer from worrying about business logic.



This month's focus is, obviously enough, on wireless and Web services. I hope you enjoy our coverage of how to mix these two exciting technologies. It's a wireless, wireless world.

WebServices JOURNAL .NET J2EE XML

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

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

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

ASSISTANT EDITOR: LIN GOETZ

WRITERS IN THIS ISSUE
BRIAN BARBASH, JOHN CANOSA, WYN EASTON, LARRY ELLINGSON, DEREK FERGUSON, KYLE GABHART, JASON GORDON, MASON HAM, GREG HEIDEL, KAM LEE, ANNE THOMAS MANES, NORBERT MIKULA, TIMOTHY OLSON, SIMON PHIPPS, SEAN RHODY, MICHAEL A. SICK, HUI ZHANG

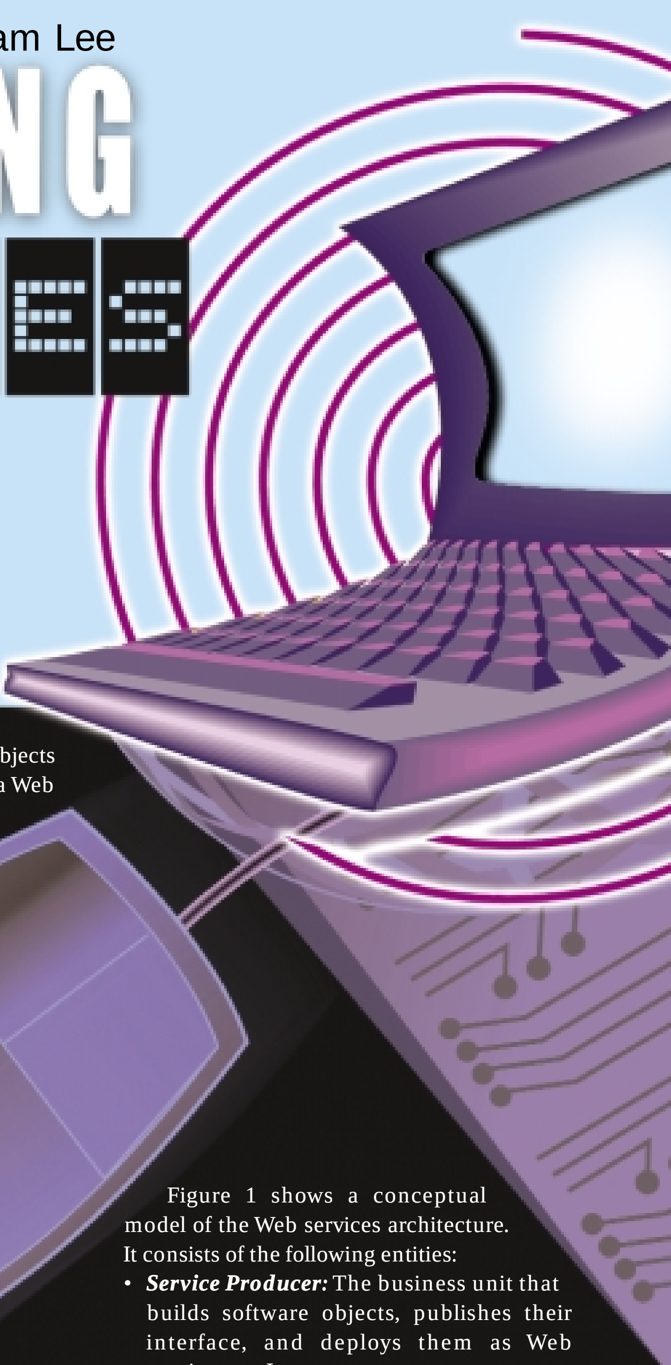
PUBLISHER, PRESIDENT AND CEO: FUAT A. KIRCAALI
VICE PRESIDENT, PRODUCTION & DESIGN: JIM MORGAN
SENIOR VP, SALES & MARKETING: CARMEN GONZALEZ
VP, SALES & MARKETING: MILES SILVERMAN
VP, EVENTS: CATHY WALTERS
VP, BUSINESS DEVELOPMENT: GRISHA DAVIDA
CHIEF FINANCIAL OFFICER: BRUCE KANNER
ASSISTANT CONTROLLER: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

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

EDITORIAL OFFICES
SYS-CON PUBLICATIONS, INC.
135 CHESTNUT RIDGE ROAD, MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 FAX: 201 782-9637
WEB SERVICES JOURNAL (ISSN# 1535-6906)
IS PUBLISHED MONTHLY (12 TIMES A YEAR) BY SYS-CON PUBLICATIONS, INC., 135 CHESTNUT RIDGE ROAD, MONTVALE, NJ 07645
PERIODICALS POSTAGE PENDING
POSTMASTER: SEND ADDRESS CHANGES TO:
WEB SERVICES JOURNAL, SYS-CON PUBLICATIONS, INC.
135 CHESTNUT RIDGE ROAD, MONTVALE, NJ 07645

written by Kam Lee

PROGRAMMING WEB SERVICES WITH SCRIPTING LANGUAGES



Web services is an emerging Internet programming paradigm that enables the remote invocation of software objects among heterogeneous systems over the Internet. There are two sides to Web services programming. On one side is the creation and deployment of Web service objects onto Internet servers; on the other side is the consumption of Web service objects in client programs or server applications. Due to the language-neutrality of the Web services paradigm, Web services can be created or consumed with different kinds of programming languages, including scripting languages such as Perl, Python, Tcl, and PHP. In this article, I'll explain and demonstrate the use of scripting to rapidly create and readily consume Web services over the Internet.

Web Services Background

Generally, the concept of "Web services" represents an enabling architecture for wide-area distributed computing among heterogeneous systems over the Internet. In this architecture, a program running on one machine can directly invoke methods in software objects running on other machines over the Internet. These software objects are called Web services. Specifically, a Web service is a collection of data and logic packaged as a single entity with defined interfaces to methods supporting data access and logic invocation. What does "Web services" mean to programmers? To server-side software developers, it means that business data and logic can be conveniently published to the network as a service for remote access and invocation by client-side programs. For example, a bank can publish a software object to supply real-time exchange-rate information to end users' financial software. To client-side application programmers, it means that additional functionality can be integrated into application programs by programmatically accessing data and invoking logic exposed on remote software objects. For example, an expense report tool can make use of the bank's exchange-rate object to convert monetary values from US$ to CDN$.

Figure 1 shows a conceptual model of the Web services architecture. It consists of the following entities:

- ***Service Producer:*** The business unit that builds software objects, publishes their interface, and deploys them as Web services on Internet servers
- ***Service Consumer:*** The end user who runs software programs that remotely access data and invoke logic from Web services launched by service producers
- ***Service Registry:*** The middleman that maintains an online directory for service producers to list and advertise their Web services and for service consumers to discover and look up Web services

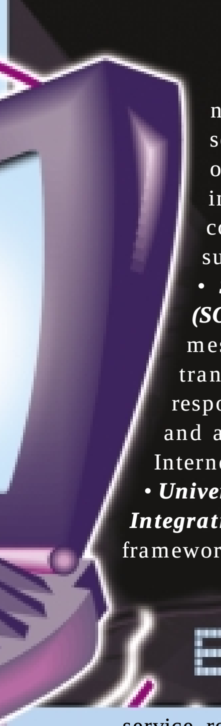
Underlying the Web services architecture is a set of well-defined open standards that facilitate communications and interactions among the entities. These standards include:

- ***Web Services Definition Language*** (**WSDL**): An XML-based language, WSDL provides a structured way for service producers to describe the functionality, specify the interfaces, and identify the

Author Bio:

Kam Lee worked at ActiveState Corp. as the technical lead for Web services initiatives. He championed product development for Web services technology. Previously, Kam was a senior designer in Microsoft's Windows Networking team and was a scientific staffer for Bell Northern Research. He has a BS in engineering physics, an MS in electrical engineering from the University of British Columbia, and a PhD in electrical and computer engineering from Carnegie Mellon University.

KAMLEESYS@HOTMAIL.COM



network address of their Web services. The WSDL description of a Web service contains all the information needed by a service consumer to invoke the methods supported by the Web service.

- ***Simple Object Access Protocol (SOAP):*** Also XML-based, SOAP is a messaging protocol designed to transport service requests and responses between a client program and a Web service object across the Internet.
- ***Universal Description, Discovery, and Integration (UDDI):*** UDDI defines a framework to enable the establishment of service registries to provide distributed directory service to the producers and consumers of Web services. It includes a common set of SOAP-based APIs to support registration and advertisement of Web services by service producers, and to facilitate the searching and lookup of Web services by service consumers.

In our example, the bank that builds and deploys the exchange-rate object acts as the service producer. It's responsible for generating the proper WSDL file and registering the Web service at an UDDI server – the service registry. The expense report tool that consumes the bank's exchange-rate object is the service consumer. The programmer of this tool queries the UDDI server to discover the service, retrieves the WSDL file, and extracts information required to invoke the Web service over SOAP.

Programming Web Services

There are two veins in the development of Web services software. First, service producers have to build and deploy software objects as Web services. We call this *service creation*. Second, service consumers have to write code in application programs to generate SOAP requests and process SOAP responses to and from Web services. We call this *service consumption*.

Web services is language-neutral, meaning that server objects can be written in any programming language, independent of the implementation of the client programs, and, vice versa, client programs can be written in any languages independent of the implementation of the server objects. So programmers of Web services have quite a few choices when it comes to selecting a programming language to create or consume Web services. On one end of the spectrum is the family of "low-level" languages such as C++ and Java. On the other end of the spectrum is the family of high-level scripting languages such as Perl, Python, Tcl, Ruby, and even PHP.

Unlike C++ and Java, scripting languages are either weakly typed or type-less, with little or no provision for complex data structures. Besides, scripts are usually interpreted rather than compiled. As such, scripting codes might not execute as fast as compiled codes. Yet scripting languages have the following advantages:

- ***Ease of use:*** The learning curve is gentle and scripts are easy to debug.
- ***Rapid implementation:*** Scripts take much less time to write and debug than compiled programs.
- ***Compact code:*** A single line of scripting code often performs the work of many lines of low-level codes.
- ***Portability:*** Scripts can often be run on different platforms with no modification or minimal porting effort.

EASY STEPS, ELEGANT METHODS

Traditionally, scripting languages have been most useful for text processing, data extraction, and file manipulation, as well as automating administrative system tasks on UNIX platforms. With the advent of the Internet, a majority of the CGI codes on Web servers are implemented in scripting languages, taking advantage of their rich set of library functions for database access and system services. Statistically, more than 50% of the Internet's Web pages are dynamically generated by Perl scripts.

There are good reasons for adopting scripting languages in the creation and consumption of Web services. Most compellingly, Web services software can be programmed, tested, debugged, and prototyped much more quickly with scripting languages than with system languages. This helps shorten the development cycle and speeds up time-to-deploy. On the consumption side, scripting can be used as a macrolanguage to integrate and glue together Web services objects in application programs. On the creation side, there exists a huge installed base of server-side Web scripts that can be redeployed as Web services with minor changes. Finally, since scripting languages are often the "native-tongue" of the current generation of Internet programmers, it will be natural for them to use scripting when it comes to programming Web services.

Creating Web Services

The creation and deployment of script-based Web services normally entails the following steps:

Step 1: **Create the implementation file**

Each Web service will be implemented in a single file containing the source code for the data and methods supported by the service. In Perl, the implementation file is nothing more than a package of subroutines; in Python, it's simply a module of functions. To illustrate, let's consider the creation of a Web service called "Advogato_Query." This Web service provides information based on content retrieved from www.advogato.com, a journalistic Web site for open-source developers. Listings 1 and 2 show skeletons of "Advogato_Query" written in Perl and in Python. (The code for this article can be found on the Web site at www.sys-con.com/webservices/sourcec.cfm.) Three methods are supported by "Advogato_Query." They are:

1. ***GetCurrentArticles()***: Returns the titles of the top five current articles
2. ***GetArticleDetails(article_title)***: Returns the content of an article titled article_title
3. ***GetArticleList(start_date, end_date)***: Returns the list of articles posted between start_date and end_date, inclusively

Step 2: **Provide the interface definition**

Because scripting languages are weakly typed, it's necessary to accompany a Web service script with an interface definition. Otherwise, the SOAP server wouldn't be able to properly and accurately perform checking, matching, and conversion of data types on the input/output parameters during invocation of the script. The interface definition should contain the names of all the exposed methods, plus the names and types of the input and output arguments for each method. Other SOAP-related metadata, such as namespace information, may be declared as well. Currently, there's no standard interface definition language (IDL) for script-based objects. As an example, consider the IDL used by the SOAP server built in PerlEx, a Perl Accelerator available from ActiveState Corp. PerlEx's SOAP IDL, called Universal Interface Specification (UIS), has a C-like syntax. Using UIS, you can define the three interfaces of the "Advogato_Query" Web service as follows:

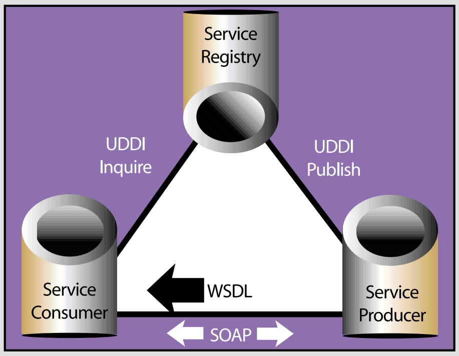


FIGURE 1 Web services architectural model

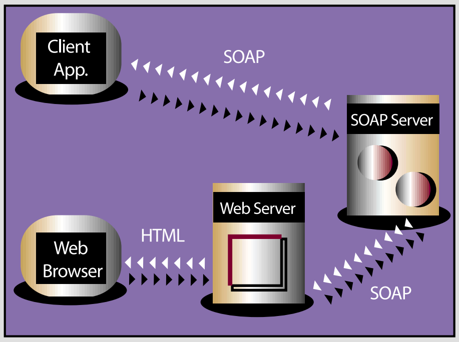


FIGURE 2 Two ways of consuming Web services

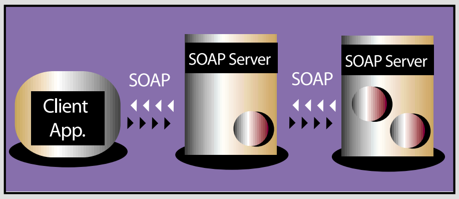


FIGURE 3 A compound Web services

```
struct dateStruct {
    int32 year;
    int32 month;
    int32 day;
}

def str_array = ustring[];

interface Advogato_Query {
    static str_array
GetCurrentArticles();
    static ustring GetArticleDetails(
ustring article_title );
    static str_array GetArticleList(
dateStruct start_date, dateStruct
end_date );
};

<soap soapaction="urn:xyz"/>
<soap namespace="http://www.efg.com/"/>
```

For the PerlEx SOAP server, the interface definition is either prefixed to the implementation file of the Web service or supplied in a separate description file.

***Step 3:* Deploy the Web service**

This refers to the installation of the Web service script on a SOAP server. The procedures are server-specific. In the case of the PerlEx SOAP server, a Web service is deployed by manually copying its implementation file (and interface definition file, if applicable) to a designated directory on the server. Other systems might support remote deployment such that scripts can be downloaded to the SOAP server over the Web.

***Step 4:* Generate the WSDL file**

A WSDL file should be created for each deployed Web service script. It can be written manually by using a text editor or XML editor. Software tools also exist that can be used to automatically generate a large part of the WSDL file based on the script's interface definition. The WSDL file for "Advogato_ Query" shown in Listing 3 indicates that the service is hosted at webservices.activestate .com.

***Step 5:* Register the Web service**

To make a newly deployed Web service known to the community, the service producer should register it with an online Web service directory. Currently, several such directories are available, including www.xmethods.com and www.webservices.org. Service registration at these portals is accomplished by completing and submitting an online registration form. For example, www.xmethods.com's registration form is available from www.xme thods.com/service. It requires the registrant to provide certain information, including service description, SOAP endpoint URL, method names, and WSDL URL. In the future, when UDDI becomes standardized and widely deployed, new Web service listings will be entered into UDDI registries via a standard process.

This completes the process of service creation using scripting languages. Let's see how a deployed Web service can be consumed by scripting codes over the Internet.

Consuming Web Services

As depicted in Figure 2, Web services can be consumed by (1) client-side applications in a two-tier architecture or (2) server-side programs in a three-tier architecture. For instance, the exchange-rate service mentioned earlier can be utilized in GUI applications designed to do financial calculations, or it can be invoked by CGI programs running on online merchants' Web servers.

For the purpose of our discussion, suppose we want to add multilanguage capability into a Perl- or Python-based lightweight e-mail client (e.g., Pail [http://pail.sourceforge.net/], Pmail [www.scottbender.net/pmail]) so that it can display the e-mail message in a foreign language. We can take advantage of Web services to realize this functionality.

***Step 1:* Discover Web services**

First we have to find and identify a Web service that meets our application's requirements. One way to do service discovery is to simply do a keyword-based search using a general-purpose search engine such as www.google. com. A better way is to browse through the listings or query the directories maintained by several Web services portals, including www.xmethods.com and www.webservices.org. In the future, when UDDI infrastructures become available, service discovery can be done more systematically and efficiently by using a UDDI browser that directly queries UDDI registries. In fact, several experimental UDDI registries have already been launched for testing and trial purposes (two examples: http://test.uddi.microsoft.com and www-3.ibm.com/services/uddi/testreg istry/ protect/registry.html).

For now, if we point our Web browser to www.xmethods.com, it will display a table of Web services available for public use. Clicking on any one of the entries in the table will bring us to the Web service's main page, which contains a description of the service as well as a pointer to the service's WSDL file. One of the services listed in the table is called "BabelFish." Driven by AltaVista's BabelFish translation engine, this Web service supports a method that translates an input text string from one language to another. It's exactly what our multilanguage e-mail client can use!

***Step 2:* Extract call parameters**

The next step is to determine the programming parameters needed to invoke the Web service. These parameters are:

- Method names
- Name and type of the input/output argument for each methods
- SOAP endpoint URL
- SOAP namespace
- Soapaction

Usually they are specified on the Web service's main page. If not, they can be extracted directly from the WSDL description of the Web service. For example, from the WSDL file of "BabelFish" (www.xmethods.net/sd/2001/BabelFishService.wsdl), we can obtain the following information:

- Method Name: BabelFish
- Input 1: name = translationmode, type = string
- Input 2: name = sourcedata, type = string
  Output: name = return, type string

SpiritSoft

www.spiritsoft.net/downloads

.xmethods.net:80/perl/soaplite.cgi
- SOAP namespace = urn:xmethodsBabel Fish
- Soapaction = "urn:xmethodsBabelFish# BabelFish"

***Step 3:* Invoke Web services**

Now we're ready to do the coding. There are three approaches to programmatically consuming Web services in scripting languages. In the first approach, native code is written to assemble the SOAP requests and transmit the request to the SOAP server and then to receive the SOAP responses and parse the responses to retrieve the results. This is called "protocol-level programming." For example, we can write a function, called translate(), to invoke "BabelFish" in Perl as seen in Listing 4. To do the same in Python, see Listing 5.

Coding at this level requires the programmer to have a good know-how of the SOAP specifications, and be proficient in network programming. Besides, the process can be tedious and error-prone, especially when dealing with array and complex data types.

The second approach is called *library-assisted programming*. It relies on the use of an external SOAP library to handle the low-level tasks of marshalling and unmarshalling SOAP calls. Table 1 lists some of the publicly available SOAP libraries for various scripting languages. Using the SOAP::Lite package, we can rewrite the translate() subroutine in Perl as:

```
sub translate
{
    my $mode = shift;
    my $text = shift;

    use SOAP::Lite
    on_action => sub {sprintf
'"urn:xmethodsBabelFish#BabelFish"'},
    uri => 'urn:xmethodsBabelFish',
    proxy => 'http://services.xmeth-
ods.net:80/perl/soaplite.cgi';

    my $dat1 = SOAP::Data ->
name('translationmode' => $mode);
    my $dat2 = SOAP::Data
->name('sourcedata' => $text);
    my $interface = new SOAP::Lite;

    my $translated_text = $interface->
BabelFish($dat1, $dat2)->result;

    return $translated_text;
}
```

Similarly, we may use the soaplib.py library to invoke the "BabelFish" service in Python (see Listing 6).

Library-assisted programming saves programmers from the trouble of assembling and disassembling SOAP messages. The task of invoking a Web service method is reduced to making several library calls.

Further simplification is achieved with the third approach, *proxy-assisted programming*. In this approach, Web services are invoked with the help of an intelligent proxy module that has built-in WSDL parsing capability. Effectively, the proxy module provides a wrapper layer above the SOAP library to completely hide the under-the-hood SOAP mechanics from the programmer. It also supports an object-oriented syntax so that programs can invoke remote Web services as if they are local objects. ActiveState's "WebService" module, available soon for Perl and Python, is an example of an intelligent proxy module for Web services invocation. Also in this category is the PHP-based SOAP4X toolkit, downloadable from http://dietrich.ganx4. com/soapx4/.

To invoke a Web service, a program only needs to supply a WSDL pointer to the proxy module's object factory function. This will create a proxy object that automatically inherits the interfaces of the Web service, and is directly callable by the program based on the methods and arguments specified in the WSDL file. For example, using ActiveState's forthcoming WebService module, we can implement the translate() subroutine in Perl as seen in Listing 7. In Python, the code will be written as shown in Listing 8.

Besides ease of use and code compactness, proxy modules may provide built-in support for call timeout, asynchronous invocation, and exception handling. When a program calls a Web service method asynchronously, the call returns immediately so that the program can perform other functions while the SOAP transaction is processed in the background. When the SOAP response arrives, the program will be notified to retrieve the result. This is especially useful in multitasking or GUI-based programs that cannot tolerate blocking. It saves the programmers from the unenviable task of spawning separate threads or forking separate processes to handle Web services requests and responses.

TABLE 1: SOAP libraries for scripting languages

Scripting Language	Soap Library	Link
Perl	Soap::Lite	www.soaplite.com
Python	SOAP.py	http://sourceforge.net/projects/pywebsvcs
	SOAPlib.py	www.pythonware.com/products/soap/
Tcl	TclSOAP	http://tclsoap.sourceforge.net/TclSOAP.html
Ruby	SOAP4R	www.jin.gr.jp/~nahi/Ruby/SOAP4R/RELEASE_en.html

Putting It Together

To tie all the concepts together, let's consider the creation of a compound Web service. Unlike the basic Web services we've examined so far, a compound Web service is itself a consumer of other Web services (see Figure 3).

Our compound Web service is called "Advogato_In_French." It supports a single method that returns a list of Advogato headlines in translated text. Listing 9 shows the Perl implementation of this Web service.

At the top of the script is the UIS-based interface definition of the service. The code makes use of ActiveState's WebService proxy module to invoke two Web Services. From the first Web service ("Advogato_Query"), it obtains the Advogato headlines in English. The second Web service ("BabelFish") is called upon to translate the headlines from English into French. Shown below is a simple Python script written to call this Perl-based Web service.

```
import WebService

news = [];

advogato_in_french_service =
WebService.new(
"http://webservices.activestate.com/wsdl
/advogato_in_french.wsdl" )
news =
advogato_in_french_service.GetArticleLis
t()

for x in news:
    print x
```

Summary

In this article, we've walked through five easy steps leading to the deployment and publishing of scripts as Web services. We've also reviewed the three different approaches to invoking Web services in scripting codes. Especially simple and elegant is the method of "proxy-assisted programming." In the near future, new features in IDEs, such as Komodo, will further simplify the process of creating and consuming Web services with scripting languages. As the need for rapid development and prototyping rises, scripting will play a prominent role in both server-side and client-side Web services programming.

Sonic

Software

www.sonicsoftware.com

Written by Dr. Hui Zhang

Advanced Data Management with Web Services Using SOAP & XML Query

As Web services grow, so does the need to manage content

The momentum behind Web services is building. If you haven't heard about them by now, you've probably been living in a jungle or some remote area for the last few years. The underlying technologies of Web services are XML, HTTP, SOAP, WSDL, and UDDI. XML provides an open standard for data exchange, HTTP an open transport protocol, SOAP a remote method invocation protocol, WSDL the service description language, and UDDI a service discovery and registry.

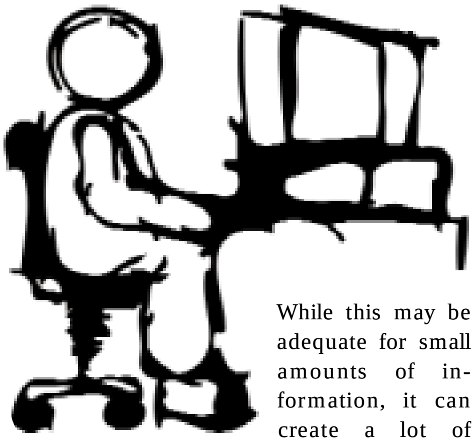
The next logical step might be creating a means to investigate and query the information available. Interestingly, Web services provides both a means for remote querying and a reason to need an XML database. As Web services grow in popularity, more and more XML content will be generated, all with a need to be managed. This article describes how to use SOAP and XML query techniques to manage the XML data stored in remote XML databases.

XML Query and XML Databases

XML has established itself as an open standard format for exchanging data and information in many areas. With the tremendous growth of XML utilization, managing XML data is fast becoming an IT headache. To manage XML data effectively, two things are needed: a query language for search and update operations, and a database to collect and organize the XML.

There are actually two XML query languages: XPath and XQuery. Both have their uses. XPath is an XML query language already standardized by the W3C. It gets its name through its use of a path notation for navigating through the hierarchical structure of an XML document. XPath operates on the abstract, logical structure of an XML document as a tree of nodes. Still in the standardization process in the W3C, XQuery is a query language that uses XML structure to express intelligent queries across all structured and semistructured XML data sources. XQuery was originally derived from the XML query language Quilt and builds on useful features incorporated from several other languages. In XQuery, a query is represented as an expression, which includes path expression (based upon XPath syntax), element constructors, For Let When Return (or FLWR, pronounced "flower") expressions, expressions involving functions and arithmetic operators, quantified expressions, conditional expressions, and expressions used for modifying or testing data types.

There are several options for storing XML data. The simplest is to use the file system.



While this may be adequate for small amounts of information, it can create a lot of overhead in searching and updating since each potential document would need to be loaded and parsed. A single transaction could span multiple documents and cause a lot of wasted time in parsing irrelevant XML. What we need is a set of indexes that describes the contents of the XML and helps to optimize the queries. Two options here are to use a relational database or a native XML database. Several of the relational database vendors provide utilities for storing XML and mapping XML queries to SQL. Native XML databases provide indexing and query capabilities optimized for XML, so we'll choose this option for our example.

There are a few commercial native XML databases in the market. For convenience, we'll use Ipedo's XML database as an example. Figure 1 illustrates the deployment model of the XML database. The database is a client/ server system. A remote client communicates with the server using client APIs. The SOAP implementation includes a SOAP client API, a SOAP client implementation, a SOAP server implementation, and the SOAP server. The SOAP server layer is basically a facade, which includes all the services you want to publish. This is the interface for the SOAP client to talk to the XML database server. We'll look at how to publish your service in the following section.

Before we start to write the code to pub-

Author Bio

Dr. Hui Zhang is a senior developer at Ipedo, Inc., where he designs and leads the integration of XML data management with Web services. His recent work includes distributed database design and streaming media.
HZHANG@IPEDO.COM

Software AG

www.softwareagusa.com/xmlweb

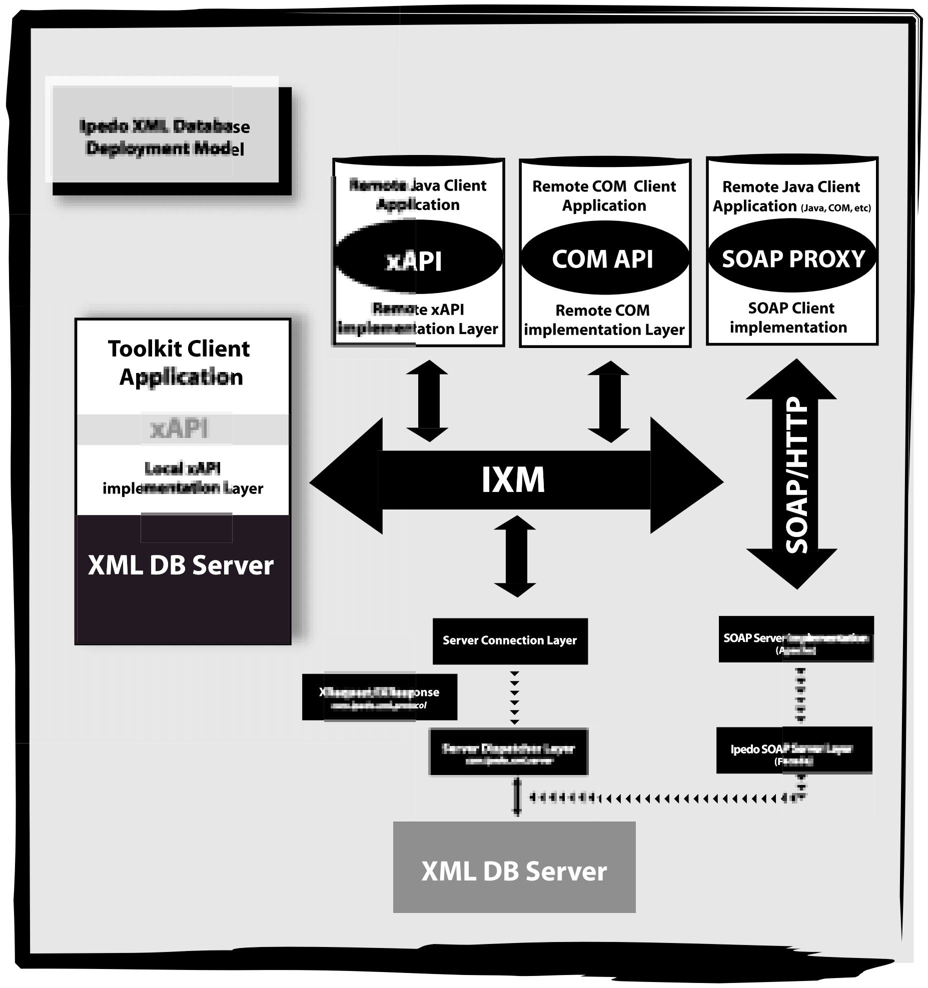

Ipedo XML Database Deployment Model
Remote Java Client Application
xAPI
Remote xAPI implementation Layer
Remote COM Client Application
COM API
Remote COM implementation Layer
Remote Java Client Application (Java, COM, etc)
SOAP PROXY
SOAP Client implementation
Toolkit Client Application
xAPI
Local xAPI implementation Layer
XML DB Server
IXM
SOAP/HTTP
Server Connection Layer
XRequest/XResponse
com.ipedo.xml.protocol
Server Dispatcher Layer
com.ipedo.xml.server
SOAP Server implementation (Apache)
Ipedo SOAP Server Layer (Facade)
XML DB Server

FIGURE 1 | XML database deployment model

lish the service, we need to understand the programming model of the XML database. Figure 2 shows the xAPI object model. A session is the entry point to all other objects. From a session, you can create database instances to which all the database management functionality is attached, and XPathStatement and XQueryStatement with which all the XPath and XQuery queries can be invoked. Transformer is used for XSLT style sheet transformation. From the database object, you can instantiate a collection object and use it to manipulate the document, adding, deleting, or updating content.

Publishing a Web Service

Now that we've covered the basics, it's time to get our hands dirty. The SOAP implementation is based on Apache SOAP open source (version 2.1). This implementation is a Java servlet, so you first need a servlet engine. We'll use Apache's Tomcat.

Writing the Server Code

I'll start with some code. The code in Listing 1 contains the facade classes with methods that are exposed to the SOAP clients (code for this article can be found at www.sys-con.com/webservices/source.cfm.)

This code allows a new XML document to be added and the document to be searched with a specified XPath query. One thing to point out here is that the SOAPServer class has no knowledge about SOAP. This means you can take your existing Java classes and expose them through SOAP.

Deploying the Service

To deploy the SOAP service with Apache SOAP, you need to define a deployment descriptor for the Java class in which it specifies several key things to a SOAP server:

- The URN of the SOAP service for clients to access
- The method or methods available to clients
- The serialization and deserialization handlers for any custom classes

The URN is similar to a URL and is required for a client to connect to any SOAP server. The second item is a list of methods letting the client know what's allowable for a SOAP client to invoke. It also lets the SOAP server know what requests to accept. The third item is a means of letting the SOAP server know how to handle any custom parameters. To simplify our implementation, we use String for all parameters.

Listing 2 shows the deployment descriptor for the SOAP server. The URN for the service is supplied in the id attribute. This needs to be unique across services, and descriptive of the service. A list of methods names is specified in the provider section as the attribute. The Java element specifies the class to expose, including its package name (through the class attribute), and indicates that the methods being exposed were not static ones (through the static attribute). Next, a fault listener implementation is specified. Apache's SOAP implementation provides two; the first one, DOMFault Listener, is used here. This listener returns any exception and fault information through an additional DOM element in the response to the client. The other fault listener implementation is org.apache. soap.server. ExceptionFaultListener.

At this point, we have both the server code and deployment descriptor ready for deployment. Now we can deploy our service. Apache SOAP comes with a utility to deploy the service. Assume that the XML database is already set up. The next thing we need to do is make the classes for our service available to the SOAP server. The best way to do this is to JAR up the service class from the previous section:

```
C:\>jar cvf SOAPServer.jar
SOAPServer.class
```

And then drop it into the lib/ of the tomcat directory and restart your Tomcat server. Use Apache SOAP's org.apache.soap.server.Service Manager utility class:

```
C:\>java
```

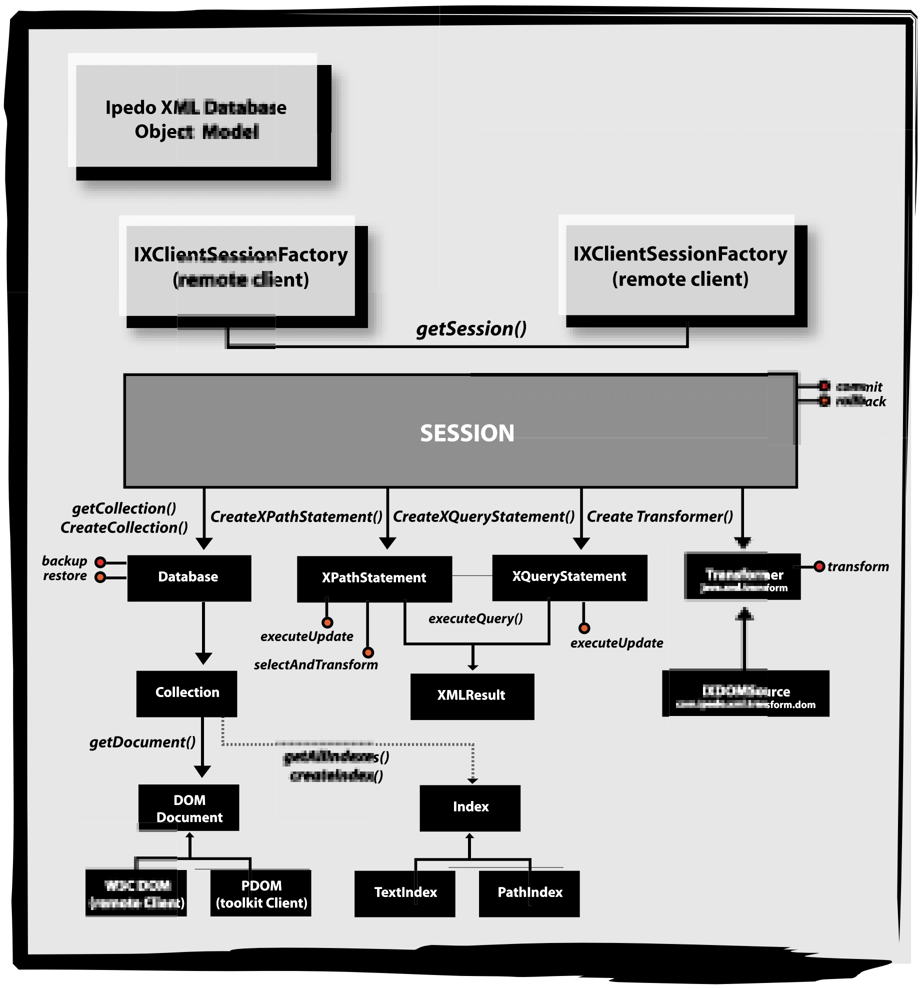


FIGURE 2 | XML database xAPI Object Model

```
org.apache.soap.server.ServiceManagerCli
ent
http://localhost:8080/soap/servlet/rpcro
uter deploy SOAPServerDD.xml
```

Three arguments are provided: the first is the SOAP server endpoint. Here it's my local machine; it could be anywhere else on the Internet. The second is the action to take, and the third is the deployment descriptor file. Once this has executed, to verify the service was added:

```
C:\> java
org.apache.soap.server.ServiceManagerCli
ent
http://localhost:8080/soap/servlet/rpcro
uter list

Deployed Services:
        urn:SOAPServer
```

Now we've published our service. Then you need to provide a WSDL file for your client to be aware of your service specification (see Listing 3).

Querying a Remote XML Database with SOAP and XPath

The next task is to write up a client to add a document and query the XML database using SOAP. If you are using a toolkit such as the Idoox SOAP wizard integrated with Borland JBuilder, based on the WSDL file of SOAP Server, you can generate the SOAP client proxy, which you can directly use it in your client application. In Listing 4 I demonstrate how to write these proxies using the Apache SOAP implementation.

Listing 4 creates proxies for the server class methods listed in Listing 1: addDocument, executeQuery, and appendChild. Within each proxy method, after setting all the input parameters, we call a method invoke, which is a wrapper for the Apache SOAP client APIs.

The main program shows how to add an XML document and then do a query against it; it takes five parameters from the command line. The first parameter is the SOAP server endpoint, the second is the name of the collection to which the document is added, the third is the assigned name of the XML document in the database, the fourth is the XML file name, and the last is the XPath query string. For example, to add a document shown in Listing 5, type:

```
C:\>java SOAPClientProxy http://local-
host:8080/soap/servlet/rpcrouter bib
bib.xml c:\xmldata\bib.xml /bib/ven-
dor/book/author[lastname='Kaufman']
```

The result will look like:

```
Document bib.xml added, docID = 8
Result of query
/bib/vendor/book/author[lastname='Kaufma
n']:

<?xml version="1.0" encoding="UTF-8"?>
<XMLQueryResult>
<author>
         <firstname>Lar</firstname>

<lastname>Kaufman</lastname>
  </author>
</XMLQueryResult>
```

Conclusion

You can do much more with SOAP and XML query. The examples shown here are relatively simple ones. What we discussed are the juicy bits of SOAP and XPath that let you create a remote query method for XML.

The fact that the query method is based on SOAP and a standard XML query language like XPath means that they can be reused with almost any XML database. They can also be used for a range of applications – everything from a catalog query application to a wireless customization engine pulling for multiple sources. The W3C is near completion of XQuery, enriching XML query techniques just in time for the explosion of XML content being driven by Web services.

Written by Norbert Mikula

(Wireless) Web Services

Only a piece of the puzzle

Why Bother?

When I was asked to focus this month's column on the subject of wireless and Web services, my immediate reaction was "Why bother?"

If you follow industry press and talk to prospects and customers, it seems that the brave new world of Web services is number 11 on the list of top 10 priorities for the wireless strategies of corporate IT managers and CIOs. Bandwidth, coverage, and device proliferation (and their associated management headaches) seem to be the real issues (or at least my top three) that bother folks today, and not some great new plumbing.

Bandwidth is certainly a concern that is well-documented – and we all know that come 2.5G or, even better, 3G, all these problems will be solved – even though we've started hearing rumors and concerns that many of the networks providers won't actually achieve the fabulous throughput we'd hoped for. (Well, I wasn't really interested in streaming video on my cell phone anyway. I'd be content if I could receive my voice-mail messages while in a roaming area.)

Coverage really has two associated dimensions. One is the availability of any kind of coverage; the other is the availability of one single provider, or at least a coherent set of protocols and standards that make our applications and devices work seamlessly. Unfortunately, we can't assume full service on either.

Device proliferation introduces a real headache for IT managers. Sometimes employees purchase their latest new gadgets and then expect full-blown support from internal IT staff. In other cases, a smart sales specialist sends a company's brand new offering – as a complimentary offer – to the ruling senior vice president of one division, while another company targets that VP's counterpart in another division, and one-two-three you have two different "standards" for wireless devices that you need to deal with.

Not much work to do here for Web services, you might say. However, I believe each of these areas can get some help from Web services. They won't solve the problem, but at least they should ease some of the pains.

What it all really amounts to is the fact that we have to deal with multiple devices (each with their own form factor) that need to receive services via potentially multiple (incompatible) technologies via carriers that may offer varying levels of bandwidth. And to add one more level of complexity, ultimately we don't deliver to devices but to human end-users (remember, those entities that make software engineering harder than it really should be).

The dynamic nature of Web services will help us create services more usable for delivery to devices requiring such a multidimensional customization approach. However, I believe it will take Web services, in conjunction with other technologies and patterns developed currently under the code name "Semantic Web," to have the potential to tremendously impact the way we address these problems.

"The more we separate the individual building blocks of our applications from each other, the more adaptable and dynamic they are"

Web Services and Wireless

Separation Is Good

Since the dawn of markup languages, we have come to realize that the more we separate the individual building blocks of our applications from each other, the more adaptable and dynamic they are and the more easily we can re-assemble them at design- and run-time.

XML (or rather its parent, SGML) was the first step in separating content from display. In programming, the model-view-controller (MVC) paradigm is experiencing a renaissance as well. Renaissance because not only do we use MVC in the context of modern-day J2EE, but developers of Smalltalk-based systems have used this approach since the '80s. Web services – the separation of business components from their associated display (or to be more precise, separating a services interface from its underlying implementation as well as separating a method-call from the transport protocol over which it operates) – is yet another piece in this ongoing trend.

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

So why do we care about this from the perspective of the wireless Web? As we have said, the challenge at hand is to build applications and services that can deal with the various factors that impact the way a service is being delivered to a wireless device. Web services, because of their componentized and de-coupled nature, are one important cornerstone of these services.

One Service for Multiple-User Interface

One of the most obvious benefits of wireless Web services is the elimination of a hard-wired user interface. Each device is unique in its form factors, screen size, support of colors, font sizes, and so on. Using Web services we can develop individual style sheets for our services that reflect the constraints for a particular device, or maybe even better the bandwidth situation at hand combined with other contextual factors (more about this later, however). When I say style sheets, I really mean any kind of technology that allows us to bind a user-interface to a business object at run-time, whether that is XSLT, JSP, or ASP conceptually doesn't matter.

From Fat-Client to (Very) Thin-Client

Web services allow us to take monolithic applications and offer discrete components of these as standards-based services. This simple step alone brings with it enormous results. Instead of potentially having to install fat clients on our wireless devices (as in the case of a PDA), we can deliver a service dynamically and on-demand. The smaller footprint of a wireless Web service also means that our dependency on high-performance networks is reduced significantly (yet not fully eliminated). As a secondary effect of wireless Web services, we may also want to mention that since we don't have to install any client software, we're able to reduce many of the management nightmares associated with these.

Having said this, what does not go away is the requirement to develop different style sheets for different devices, or at least categories of devices, as required. The difference is that, instead of having to install an update on the client device each time we change something, we can perform these changes on the server-side and, automatically, the next time the client tries to access that service, the new style sheet will be used.

"**We need to start capturing** enough meta-information about our information assets **so that we can do** more intelligent work with them"

(Semantic) Wireless Web?

Having Web services that provide us with an agile business backbone that's highly dynamic, parametric, and customizable is great. However, without information about how we utilize this flexibility, it's almost like a car engine without wheels. While this is true for any device, it's even more important for wireless devices, because of the complexity of the service delivery issues involved.

I am a firm believer that the power of the wireless Web will largely depend on our ability to deliver on the vision of the semantic Web. There are a lot of discussions underway about the semantic Web and you can easily get lost in academic thoughts (I will, for now, try to steer clear of discussing the beauty of ontologies, as I don't want to create major headaches for you and myself).

To keep it simple, in my mind, the semantic Web has fundamentally three cornerstones: information, devices, and users.

Three Cornerstones of the Semantic Web

Information: The semantic Web (unlike the current Web) is an infrastructure where computers and humans are equal citizens. As such, we need to store and capture information so that humans and machines alike can process it. To some extent this means describing data using standards like XML, but also providing additional explicit information about that data by means of metadata (maybe using RDF). This applies as well to documents that aren't in XML form, as in various kinds of electronic office documents. Why? Because only if we know enough about the information we (the citizens of the semantic Web) are processing can we do things like a high-precision search that really delivers the results we are looking for. Only with enough metainformation can we help machines differentiate between the meanings of certain words used in different contexts. Conversely, we also need to enable machines to understand that FirstName, Fname and FN are really the same concept, the first name of a person, even if their labels differ.

Devices: As we've discussed in previous sections, each wireless device has its own profile associated with it. These profiles need to be captured and made available in a standards-based fashion. Some of the characteristics of a device profile will be static (such as the often-cited screen size); others will be more dynamic in nature, bandwidth is a prime example.

Users: We need to capture information concerning the ultimate target of any information processing – again, the human end-user. The who, what, how, where, and why of calling a service – from a user perspective – will ultimately determine in which form this service needs to be rendered to the user.

Who, How, What, Where, and Why

Who is the user that requests a service? This is the first crucial question for any services delivery, wireless included. The "who" will determine if a user is permitted to enter our system – let's say an enterprise portal to begin with. However, we can go beyond these basic security issues and also determine any kind of preferences a user might have regarding their wireless service. Do they prefer a certain color-scheme, a certain font size or a certain way of ordering search results after a query? The more we know about who a user is and what their preferences are, the better we can cater to these individual choices.

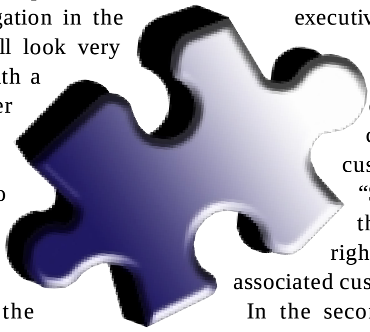
How a service is being accessed includes information about a device and its associated profile. Are we talking about a browser on a laptop, a browser on a PDA, a cell phone, or a voice interface? Whatever the answer to this question is will determine the way a service needs to be presented to the user.

What is being accessed? Once we've authenticated the user (i.e., once we have made sure a user is the person they claim to be), we can determine whether a user is entitled to access that service to begin with. In addition, through looking at the *what,* we may need to add additional processing steps, such

"We need to develop Web services so that they are flexible enough to support the various interaction modalities required by the wireless devices"

as converting a document to be suitable for the form factor of the device at hand. Beyond simple data and document transformation, depending on the device we may choose to use a different strategy to allow users to navigate between the various service components. For example, strategies for navigation in the context of a cell phone will look very different than, say, a PDA with a Web browser and a wider screen. If I access a document I may need to split it into segments requiring the user to press a button to page down; on a PDA we may be content with taking the whole document and relying on the installed browsers' scrolling capabilities (we still may have to convert heavy graphics to some smaller and more digestible format).

The *where* question takes us into the brave new world of location-based services. Let's say you call on the weather service. A device should be intelligent enough to know that you are in Chicago and not ask you to supply a ZIP code. But what if a device doesn't have location information available? There are numerous clues from which we can deduce the location of a user. How about checking the electronic calendar? It's Thursday; the electronic calendar says the user is (supposed to be) in Chicago – ergo? (Before being accused of smoking illegal substances, I do readily admit that to deliver on that vision a few other things have to happen as well, but that is exactly what the Semantic Web efforts are starting to do.)



Why? Now it gets really tricky. Actually, there's less black magic involved than you might think. As an example, let's look at a customer relationship management service. Imagine two scenarios: in the first one you are the sales executive and you're at a customer site. You start the service via your PDA, it asks for a customer name and launches a search. It finds 10 customers by the last name of "Smith," and after paging through the list you find the right one and bring up the associated customer record.

In the second scenario, however, you again start by launching the service; the service, however, knows (thanks to structured information in your office system) that you are supposed to be at Smith's office at this time of the day and after confirming the fact that this is still the case (you may be running late), it displays the customer record for you. Which one would you prefer?

A Slam Dunk?

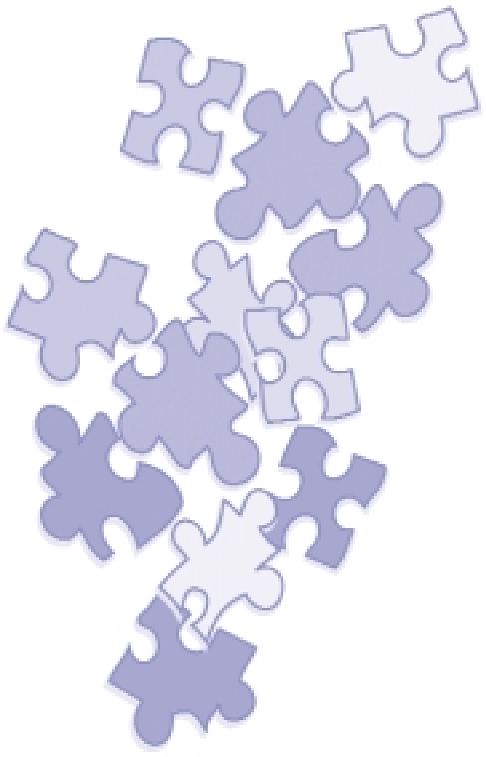
In the previous sections I've presented a vision that combines the wireless Web with Web services as well as the semantic Web. But none of the glorious outcomes described will be achieved for free.

We need to start capturing enough meta-information about our information assets so we can do more intelligent work with them. This would be almost impossible to do manually. However, I believe increasingly (semi-) automated tools will allows us to analyze our structured and unstructured data-sources to help with the creation of more intelligent metadata.

We need to develop Web services so that they are flexible enough to support the various interaction modalities required by wireless devices. It will be interesting to see which patterns for business objects and style sheets we will see emerge over the next months.

Most important, we'll need to put the user at the center of what we do. The e-business systems we build need to understand that whatever a user does, it does for a particular reason. Only if our systems are smart enough to understand a user's context for an action can we optimize a service for a wireless device (or any device for that matter).

Some of this context we can derive from explicit sources such as a business process description or a workflow. On the other hand, we'll be able to derive a lot of the context from a user's behavioral patterns and actions. In many ways, the time has come to use the "reasoning" power of computers to help services understand what we do and why we do it. The rise and fall of a certain paper clip, friendly and always eager to help, reminds us how difficult this task can be.



"Only if our systems are smart enough to understand a user's context for an action can we optimize a service for a wireless device"

Borland

www.borland.com/new/k2/2014.html

BRINGING WEB SERVICES to Smart Devices

written by John Canosa

Realizing the value of the connected world

Author Bio:

John Canosa is chief scientist of Questra Corporation. He has more than 20 years of experience in the industry, including communications systems, analog circuit design, microprocessor and computer design through large software design efforts, and pretty much everything in-between. John has been a highly rated speaker at numerous conferences and has delivered training and tutorial sessions at many Fortune 500 companies.

JCANOSA@QUESTRA.COM

Imagine that Web services becomes successful beyond the wildest dreams of anyone involved with the technology. Take your imagination even further and suppose that every single PC, workstation, and server on the face of the earth communicates using Web services.

While that sounds like a fantastic scenario, if it were true we'd still be at a point where significantly less than 1% of installed microprocessors would be using Web services. So where are the rest of these microprocessors? They're embedded in the intelligent devices that pervade our world. There are literally billions of microprocessors sold each year, ranging from the low-end 8- and 16-bit systems that can be had for under a dollar to the high-end 32- and 64-bit systems that perform complex processing and control algorithms. From office copiers to automobiles to manufacturing equipment, and even down to lowly air conditioning systems, intelligence in the form of one, two, or even more microprocessors is present.

It's somewhat humorous, but ultimately disappointing, that many of the proponents of pervasive computing who talk about supporting devices of all types immediately eliminate 80% of the microprocessors sold each year because of their architectural decisions. In terms of number of devices shipped, embedded 8-bit microprocessors still outsell the more powerful 32- and 64-bit systems by almost an order of magnitude. These lower-end devices collect information that's just as valuable as that collected by their more powerful brethren, and they have even more to gain by being connected to a powerful distributed computing environment.

All of these intelligent systems are capable of generating, collecting, and storing vast amounts of valuable information. Unfortunately, this information has often been trapped inside the devices because there was no way to communicate with the enterprise systems that could turn this information into actionable knowledge. Web services are a powerful mechanism for connecting distributed, intelligent assets to the business enterprise to provide such valuable services as automatically generating service requests, monitoring usage, performing remote

diagnostics, and reordering consumables.

Web Services in the Embedded World

The lines between embedded devices and enterprise software are blurring rapidly. Intelligent devices in the field contain incredibly valuable data about status, historical usage, consumables needs, wear and tear, and other parameters. In their continuing evolution (see Figure 1), connected devices have moved from standalone systems to the many Internet-enabled devices that exist today.

However, most developers have stopped short of full evolution by providing simple Web browser–based interfaces or, at best, point solutions. The drawback of these systems is that they don't get the information into the enterprise systems that can extract the tremendous value that exists. That value may expose itself as reduced field service operating costs, usage-based billing, additional revenues through replenishment systems, or some other value-added service that can be provided through the device. To do this, an end-to-end solution must connect the devices into enterprise applications such as an e-commerce, ERP, or field service application. What better way to accomplish this goal than by speaking what is becoming the native language of these enterprise applications – namely, Web services? A couple of concrete examples should clarify how an embedded device might use a Web service.

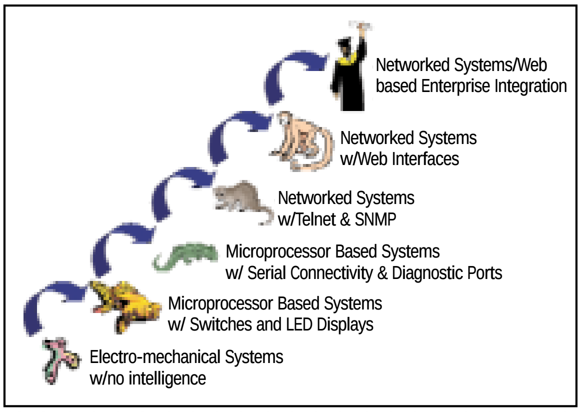

Networked Systems/Web based Enterprise Integration
Networked Systems w/Web Interfaces
Networked Systems w/Telnet & SNMP
Microprocessor Based Systems w/ Serial Connectivity & Diagnostic Ports
Microprocessor Based Systems w/ Switches and LED Displays
Electro-mechanical Systems w/no intelligence

FIGURE 1 | **Evolution of connected devices**

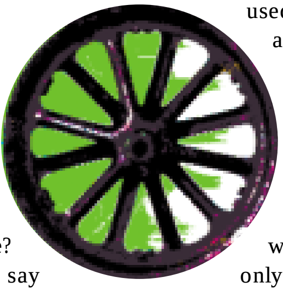
"The large number of packaged and custom-developed enterprise applications is **surpassed only by the incredible** number of Different types of embedded systems"

Accessing Web Services

Dispensing systems are good examples of critical assets. Whether they're dispensing specialty gases and chemicals for the production of integrated circuits, pumping liquids in a manufacturing process, or dispensing cleaning chemicals in a commercial kitchen, the operation of these devices is absolutely critical to the end user's business. Let's use an example where a dispensing system has detected a failure condition in one of its motors. In the prehistoric era of intelligent devices, the best the dispenser might have been able to do was write to a log file and put a message on its LCD interface to indicate that service was required. In a Web services world, the device may execute the previous two steps, but also call what appears to be a local Event Notification function. Under the hood, a Web service proxy would take the notification, wrap it up in a SOAP-encoded message, and invoke a remote Event Web service. The Event Web Service would take this event, a motor failure, and process it according to its predefined workflow and business rules.

The result of that processing may be the invocation of other Web services, such as submission of a service request into a field service application, as well as a check on the availability of a spare motor from an inventory management system. Through the use of Web services and Enterprise Application Integration, this simple request may trigger a variety of actions within the enterprise, enabling a highly automated environment. The result is that a simple component failure has automatically set into motion a complex series of events (before the customer may even be aware of the problem) that ultimately results in a faster response to a problem and a high degree of customer satisfaction. This is only one example of a Web service that could provide value to an embedded system. Ordering consumables, usage metering, and operational efficiency monitoring are just some of the functions that could be provided with the proper Web services.

“These lower-end devices collect information that’s just as valuable as that collected by their more powerful brethren”



Providing Web Services

Outbound invocation of a Web service makes sense, but what about the other direction? Why would a device want to be the provider of a Web service? To continue the previous example, let’s say that the field service application has sent a wireless notification to a field technician’s PDA. In the old days, the technician would call the customer to schedule a visit, arrive at the site, and plug into the dispenser to perform a set of diagnostic routines. All too often, this diagnostic sequence suggested that the proper fix involved a part that the technician didn’t have with him, and a trip to a warehouse was in order. In a Web services world, the technician can use his PDA to remotely invoke the diagnostic routines; in essence, the PDA is the client to the Web services offered up by the device. The technician can perform the diagnosis remotely and arrive at the customer’s site with the proper replacement parts in hand. Other functions, such as firmware upgrades and remote control, could also be provided using the Web services standards.

Furthermore, by allowing remote devices to “publish” their available services to public or private UDDI registries, the availability of services such as remote diagnostics or firmware upgrades can be determined dynamically. This provides greater flexibility and extensibility when creating applications that a typical field service technician would use. No longer does this functionality have to be hard-coded for specific device types.

Design Issues

Each new technology seems to bring with it a whole new set of design trade-offs and issues. Web services are no different. Embedded systems just seem to exacerbate the problem. Limited resources, severe cost constraints, and operational considerations combine to create a complex set of engineering trade-offs. A discussion of some of the purported issues surrounding Web services follows.

Security

Security is a major issue any time you’re exposing device information to a public network. Security has many aspects, including encryption, authentication, and authorization. Unfortunately, the SOAP specification is noticeably silent about this issue. Any serious implementation of Web services must address such issues. Luckily, SOAP allows for an XML wrapper to be placed around the actual SOAP message. Within that message framework credentials can be presented and used to authorize a specific action. Fortunately, some Web service framework implementations targeted to the embedded world support authenticated access to authorized services. This works both ways in that only authorized people can access certain Web services provided by a device, while certain Web services provided by an enterprise may only be available to authorized devices.

Complexity

Just as when object-oriented programming (OOP) started to filter down into embedded systems, detractors may say that while Web services is fine for large systems, it just doesn’t make sense for the embedded world. And, as with OOP and C++, people also tend to confuse the concept with a specific implementation. Web services is no different: it combines some of the best features of OOP and distributed computing. Let the processor best suited to a particular process or calculation handle that task but keep the actual inner workings of that task hidden behind a specific interface. Implementing the networking, XML parsing, and SOAP encapsulation can be a bit intimidating but there are tools designed to abstract away much of the complexity. For example, one company provides a WSDL compiler targeted specifically for embedded systems. Such a code generator takes an XML-based WSDL document as input and produces C++ code that implements a proxy for a Web service. For the application developer, this means that the Event Service (highlighted in the previous examples) is accessed with what appears to be a simple local function call such as Event.SubmitEvent(MotorFailure). The proxy software takes that function call, wraps it in a SOAP message, and sends it out over the network. When the response to the SOAP message arrives, it is parsed and the result code is returned to the calling routine.

This is especially attractive when taking an existing design and giving it the capability to access Web services. Most existing devices write to a log file or output a diagnostic code when a fault is detected. Submitting that fault event to an enterprise application becomes a matter of adding one more function call in the fault handling routine.

Resource Constraints

One of the main features that defines an embedded system is that you don’t have enough memory, processing power, or some other resource to do what you want to. The thought of adding an XML parser and SOAP encoding engine to a system seems problematic at best. In many cases that might be true – a full-blown XML parser can easily add over 180Kb of code. Fortunately, many of the features of the XML standard are not required for SOAP encoding. A well-pared XML parser that fully supports SOAP can be under 20 Kb in size. For those truly constrained devices that can’t handle even that small amount of code, Web services can also be invoked without using SOAP at all. The WSDL specification also allows a port to be bound to an HTTP POST or GET verb targeted at a specific address. This allows the invocation of a Web service to be as simple as sending an HTTP POST or GET URL-encoded request to a specific URL as shown below:

```
POST /EventService/EventServlet
HTTP/1.1
Content-Type: application/x-www-form-
urlencoded
Content-Length: nnnn
[CR][LF]
operation=EventService&authenticationTok
en=xXXdDj2edph
&description=motot+failure&source=54321
```

Web service frameworks targeted towards embedded systems exist that provide an HTTP client and server and support URL encoding while consuming under 5Kb of code space.

Verbosity

The verbosity of the text-oriented HTTP solution has multiple system impacts, affecting RAM usage, bandwidth requirements, and operating costs. XML is a text-based language and provides the significant advantage of making platform independence a trivial task. The downside is that text-based systems are inherently less efficient than a binary system. This leads to more data being transmitted and larger buffers required to both prepare outbound messages and receive inbound messages. One approach to ad-

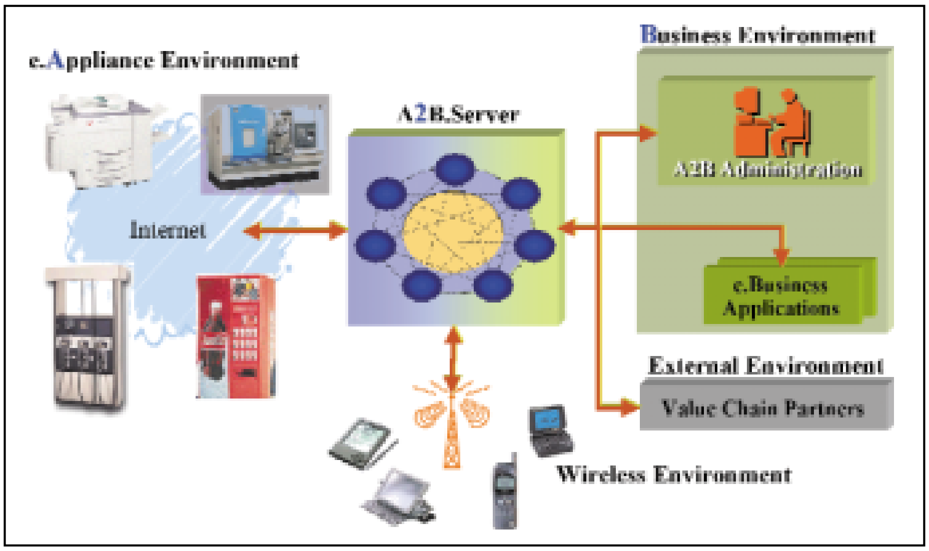


FIGURE 2 Integrating people, devices, and enterprise systems

dressing this downside is a technique called Compact URL encoding. This encoding scheme builds on the URL encoding provision built into the WSDL specification but adds further compression to minimize both RAM usage and the amount of data sent over the network. Savings of 5x to 10x are easily achievable.

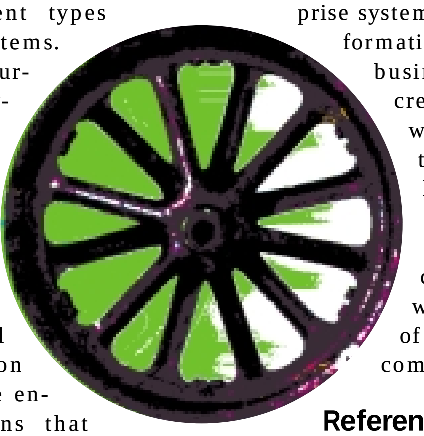
8- and 16-Bit Support

Many platforms that claim to support embedded systems really mean they support 32-bit systems that run a real-time operating system (RTOS). In other words, they support only about 15% of the processors out in the field. What about the other 85% of the processors, are they doomed to proprietary networking systems at best or standalone operation at worst? Certain proprietary networking software companies that say an 8-bit processor simply can't handle a full-blown TCP/IP stack have been proven wrong (there are at least a half-dozen 8-bit TCP/IP solutions available today). Combining these stacks with the URL encoding technique mentioned above creates a whole new opportunity for connected systems.

If the view of Web services as a distributed computing model is subscribed to, many new applications and features become possible. For example, addressing the slowly deteriorating performance of an industrial compressor can be a complex statistical calculation based on historical data and current environmental factors. If a more capable application server has access to this information it can handle the calculations, generate a service request for a field technician, and also provide feedback to the device, such as going into a lower rpm mode to minimize vibrations in order to extend the life of the asset until help arrives. This gives the end customer continued use of that asset but perhaps at a lower capacity. Eight-bit applications that can invoke two different Web services over a TCP/IP stack (with full gateway routing support) have been demonstrated in under 20 Kb of code on an 8051 type architecture. Consider the impact of even the most mundane devices, such as a compressor or smart sensor, being able to participate in the community of a business enterprise.

Enterprise Integration

The large number of packaged and custom-developed enterprise applications is surpassed only by the incredible number of different types of embedded systems. The business rules surrounding specific events and actions add even more complexity. To extract the value of the information provided by these newly connected devices, it's critical that this information be accessible to the enterprise applications that can analyze the information and apply the business rules. Standalone visualization tools such as an HP OpenView only provide a view at the current point in time; historical information – which is extremely valuable – is often lost. Other solutions that provide a database to collect historical information along with a specific point application are also available. Unfortunately, these solutions provide little or no integration into existing enterprise solutions that often cost a company several million dollars.



The different data models and APIs make this integration an onerous task. In addition, devices may want to talk to multiple applications, such as both a field service application and a spare parts procurement system, making things even more complicated. By incorporating customizable workflow engines into an enterprise's Web service environment, end-to-end solutions can be created that benefit all aspects of both the service and supply chains. As mentioned earlier, enterprise applications are quickly moving to the Web services model. What better way to ease the integration of devices into this enterprise world than to speak the language!

Summary

Web services is rapidly becoming the native language of business applications. History shows that technologies designed for the enterprise often find their way into embedded systems and that the speed of adoption for these technologies is increasing. Capturing the real value of connecting devices to the Internet goes much further than providing a standalone database for collecting information or providing some proprietary XML interface.

As shown in Figure 2, a system-level Appliance-to-Business (A2B) approach results in the connection of the devices that contribute information, the enterprise systems that can analyze the information and apply the proper business rules, and the increasingly mobile workers who can act on the results of the aforementioned analysis as well as interact remotely with the devices. Only when these three constituencies are served will the tremendous value of the connected world be completely realized.

References

1. "A2B Technology Overview," Questra Corporation, June 2001, www.questra. com.
2. "SOAP Specification V1.1," W3C Note 08, May 2000, www.w3.org/SOAP.
3. "WSDL Specification," W3C Note 15, March 2001, www.w3.org/TR/wsdl.
4. "UDDI Specification," uddi.org, September 2000, www.uddi.org.

Data mirror

www.datamirror.com/resourcecenter

SOAP on a ROPE

Requested Objects Passed Elegantly

Converting a Web site to a Web service requires some new tools in your programming toolbox. For Web services that need to exchange arbitrary Java objects, you can add the tools described here. I'll show you how to exchange Java objects using SOAP RPC and invoke methods on these objects.

The source code for the examples as well as the listings discussed here can be downloaded from www.sys-con.com/webservices/sourcec.cfm. The example code consists of a client and a simple Web service (see Figure 1). The client is a Java application named *Client*. The Web service code is a Java class called *ObjectDepot*. ObjectDepot exposes two SOAP callable methods using a deployment descriptor named *DeploymentDescriptor.xml*. The method getMyObject() will return an instance of AClass. The method getObject (String name) returns an instance of a named class that is passed as a String parameter. BClass is used in the example. AClass and BClass are simple do-nothing classes I've used for demonstration purposes. The only restriction on the objects that you send and receive is that they must implement the Serializable Interface.

The client makes a SOAP method call that returns a Java Object. The dotted lines represent the marshall() and by ObjSerializer. unmarshall() method calls made I used the following tools in addition to the Java 2 Platform, Standard Edition (J2SE) to set up my development environment:

- Apache SOAP 2.2: http://xml.apache.org/soap/index.html Note: Version 2.2 of Apache SOAP requires two additional .jar files: mail.jar and activation.jar. The mail.jar file is part of the JavaMail package and activation.jar is Part of the JavaBean Activation Framework. Both .jar files are available from http://java.sun.com.
- Apache Jakarta Tomcat 3.2.2: http://jakarta.apache.org/site/binindex.html
- Apache Xerces XML Parser 1.4.1: http://xml.apache.org/xerces-j/

If you are not familiar with Apache SOAP, there are several examples of using SOAP in the sample directory of the SOAP zip file. For this article, assume you have executed one of the SOAP Remote Procedure Call (RPC) sample programs.

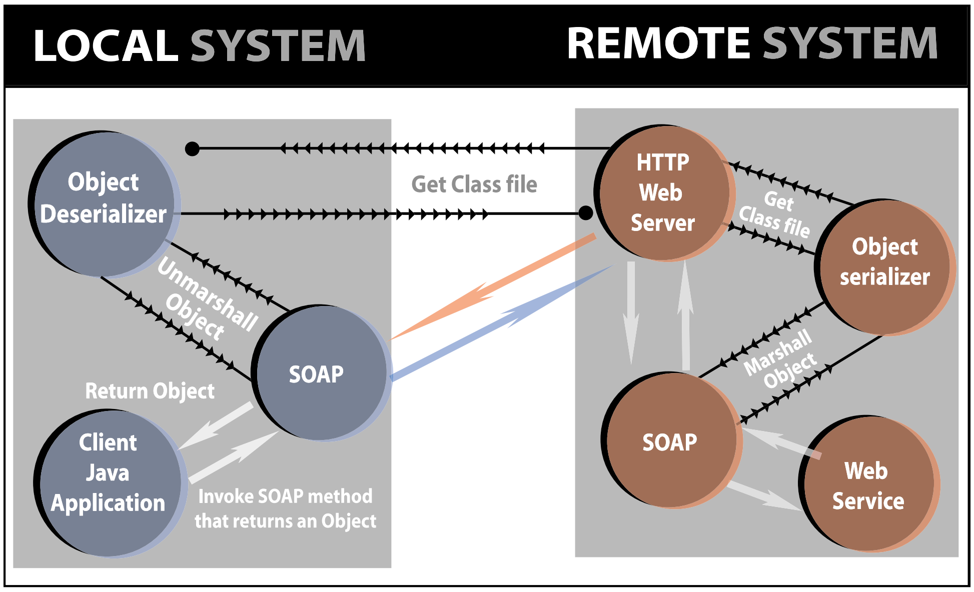


FIGURE 1 Invoking a Web service method through SOAP that returns a Java Object

A Little Background

Any type of data returned by invoking a SOAP RPC needs to be marshalled at the sender and unmarshalled at the receiver. Apache SOAP uses type mappings to determine how Java data types should be marshalled to XML so that the data can be transmitted on the network. The same is true when the data is received at a client.

Another type mapping is used to determine how the data should be unmarshalled. The type mappings are stored in a type-mapping registry. The default registry is org.apache. soap. encoding. SOAPMappingRegistry. Apache's SOAP implementation contains serializers and deserializers that will marshall and unmarshall many of the basic Java objects. Among the objects supported is the Java String Object, and there is even a Java Bean serializer and deserializer. After browsing the SOAP documentation, I wondered why any Java object couldn't be transmitted. This was in the Apache SOAP documentation:

> "If you need to create a new serializer/deserializer, then looking at the source code for the predefined ones will probably provide you with the best guidance. Remember that they will need to implement org.apache .soap. util.xml. Seri alizer and org.apa che.soap. util.xml. Deserializer respectively. You can implement them both in the same class or use two different classes. It really makes no difference."

There was my answer. I could write my own Java object serializ er/deserializer.

After digging a little further, I discovered that there are several technical hurdles to scale. First I needed to serialize the object. Then I'd need to fit the

Author Bio
Wyn Easton, an advisory software engineer with IBM, has worked with Java for four years and Web applications for two. He has a patent application for some of his work with XML and Java.
WEASTON@US.IBM.COM

Shinka Technologies

www.shinkatech.com/register7/

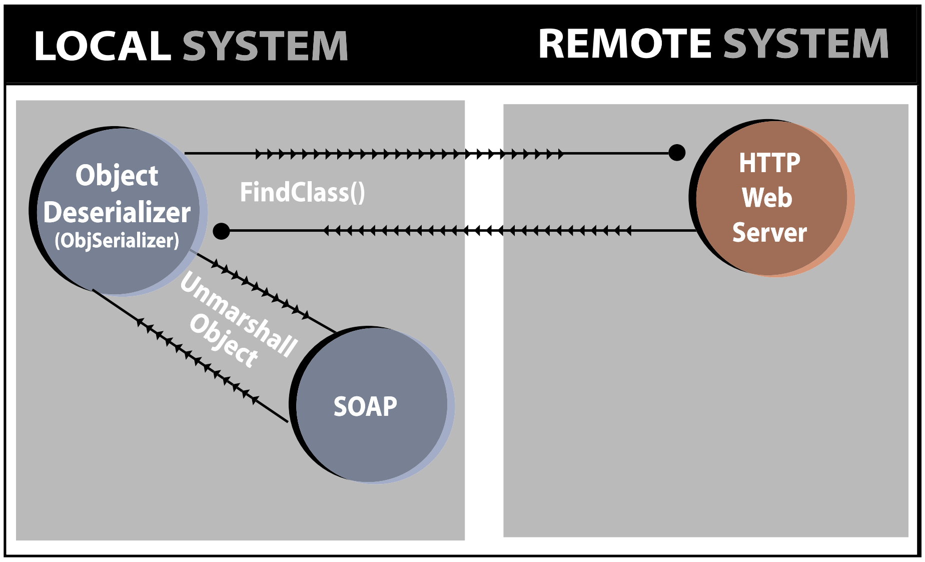


FIGURE 2 | **A SOAP Exchange**

serialized object into the SOAP envelope. Once the object reached the client, it would need to be deserialized so that it could be used. The biggest hurdle was getting access to the class file on the client so that the object could be deserialized.

My Approach

I'm sure you're aware that developers don't usually write their final version of code on the first attempt. Instead of describing the final version of my code, I'm going to describe how I arrived at that point.

To get started, I needed a serializer and deserializer for my Java objects. My implementation of both is contained in Obj Serializer.java. Objserializer implements the Apache SOAP serializer and deserializer interfaces. I used Java's standard Object Input Stream and ObjectOutputStream to do most of the heavy lifting (see Listing 1).

I also used an Apache SOAP utility called Base64 to convert the binary object data to a base64 encoded String.

```
  byte byteArray[] =
ObjectToArray(value);
  Base64.encode(byteArray, 0,
byteArray.length, sink);
```

I did this to prevent any encoding that an application server might try to do.

Wire Tapping

Once I had the serializer working, I used another Apache SOAP utility called *TcpTunnel Gui* to see what was being sent over the wire. The TcpTunnelGui utility works as a relay between the client application and the Web server. The SOAP envelope sent by the client is displayed in one half of the window and the reply is displayed in the other half. Figure 2 shows a typical SOAP exchange. I could see that my serializer was working by looking at the reply envelope. I got several complaints from the Client code, but that was what I expected since I hadn't written any code to handle the returned data yet.

As mentioned earlier, I saw that the deserializer on the client machine had to have access to the object's class file for it to be deserialized for the client to use. When I began, the only way I could accomplish this was to copy the class file from the Web server to a file in the class path of the client just before the class file was needed for unmarshalling the data into the object. I added some code to the catch block in ObjSerializer that caught the ClassNotFoundException that was being thrown by the readObject() method of ObjectIn putStream (see Listing 2).

In the catch block I retrieved the class file from the Web server and then retried the unmarshalling. After the readObj ect() method returned the object, I removed the class file from the disk. I left my original unmarshall() method in ObjSerial izer.java for you. The unmarshalling worked and I could use the returned object, but I thought there must be a more elegant way. I posted a note on the Apache SOAP mailing list explaining what I was trying to do and someone suggested I look at class loaders. After a little reading I realized that URLClass Loader would do the job and I only needed to override the findClass() method in URLClassLoader.

A Class Loader Mini-Course

The entire subject of class loaders is beyond the scope of this article. You need, however, to know about the *delegation model* to understand how my code works. A class loader has a parent class loader. The set of a class loader and its ancestors is called a *delegation*. Standard Java applications begin with a delegation of three class loaders. The system class loader loads classes from the class path, delegates to the extension class loader, and is used to load Java extensions. The parent of the extension class loader is the bootstrap class loader. The bootstrap class loader loads the core API.

When MyURLClassLoader is added to the delegation, the system class loader becomes its parent. When a class loader loads a class, it consults its parent first to give the parent a chance to load the class file. Whenever a class refers to another class, the same class loader that loaded the referencing class loads the referent. In other words, since ObjSerializer is loaded by MyURLClassLoader, MyURLClass Loader will be used to load classes needed by ObjSerializer. That is done when the system class loader calls the findClass() method of MyURLClassLoader after failing to find a class file when the readObject() method is called from ObjSerializer's unmarshall() method (see Figure 3). The findClass() method needs to know the host name of the Web server and where the class files are located on the server. These are read from ObjSerial izer.properties when MyUrlClassLoa der is instantiated. The unmarshall() method calls the readObject() method of Object Input Stream. Which then calls the find Class() method of MyURLClass Loader.

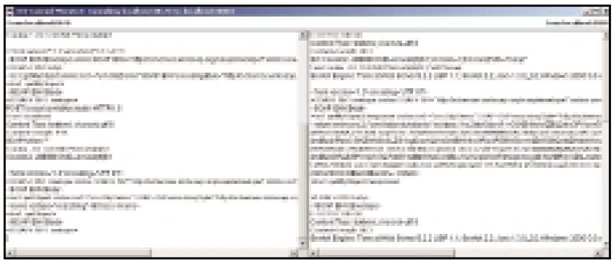
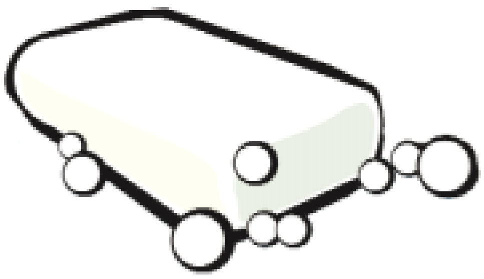


FIGURE 3 | **Class File Retrieval Sub-Process**

Infragistics

www.infragistics.com



The findClass() method retrieves the class file from the remote web server. The readObject() method returns the object to the unmarshall() method. The unmarshall() method returns the object in a SOAP Bean.

Here are the contents of my ObjSerializer. properties file:

```
ClassDefHost=localhost
ClassDefPort=8080
ClassDefDirectory=/objser
```

Listing 3 from MyURLClass Loader builds the URL for the class file when it's needed (the class file we need is called *name)*. I stored AClass. class and BClass. class in a directory named *objser* under Tomcat's webapps directory.

As I explained earlier, I would need a Class that would load ObjSerializer using the Class.forName() method. That way I could pass an instance of MyURLClass Loader to be used to load ObjSerializer. This project was getting a little complicated, but I was almost home. I created ObjSerializerLoader.java to do the job (see Listing 4). Okay, now I was ready. I had my SOAP deployment descriptor set up to register ObjSerializerLoader as my serializer and deerializer on the SOAP server. I also registered ObjSerializer Loader in Client for the same purpose. I started up Tomcat, ran Client which invoked ObjDepot's getMyObject() method through SOAP RPC.

I was upset to see a ClassNotFound Exception, just as I had seen before. It looked like MyURLClass Loader wasn't being used at all. After a few System. out.println() statements, I discovered I was right. MyURLClassLoa der wasn't being used. As it turned out, I had made only one mistake. When I used the Class.f or Name() method to create my instance of ObjSerializer, I assigned it to a variable of type ObjSerializer. Since the system class loader could find the class file for ObjSerializer, the system class loader was used to create the object and not my class loader. What I needed to do was put ObjSerializer into a variable of type Object and use Java Reflection to invoke the marshal() and unmarshall() methods (see Listing 5).

That way there would be no reference to ObjSerializer in ObjSerializerLoader, so my code would compile without ObjSe rializer in the class path and the system class loader would delegate to MyURL ClassLoader's find Class() method when ObjectSerializer is unmarshalling the returned data. As I mentioned earlier, the delegation takes place when the read Object() method is called in the unmar shall() method of ObjSerializer (see Listing 6).

"If you download the code, read the comments, and refer back to this article, you should be able to figure out how to use SOAP to transfer Java objects from your Web service and use them on a client."

If you're like me, you've spent some time figuring out class path problems when some class file couldn't be found. In this case, I didn't want ObjSerializer to be found by the system class loader.

Try, Try Again

After a few modifications, I was ready to go again. When I tried the client this time, it worked. I had sent a Java object across the network using SOAP RPC and executed that object on the client machine by using Java Reflection to invoke the main() method of the returned object. This was done without having any reference to the class file on the client machine. Life was good.

A word about security: as powerful as this tool can be, it can also be dangerous. This tool should only be used among trusted Web sites. You should also consider encryption, based on the sensitivity of the data transmitted.

Running the Example Code

Here is what you will need to do to run the example code.

- On the client workstation, create a directory called Client.
- Extract Client.java, ObjSerializer.java, Obj SerializerLoader.java, MyURLClass Load er.java, Deployment Desciptor .xml, Obj ectSerializer.properties, and wssetup.bat into the client directory.
- On the Web service workstation, which can be the same as the client workstation, create a directory called "objde pot". Extract IObjDepot.java, ObjDepot. java, Aclass.java, Bclass.java, MyURLCl assLoader.java, and wssetup.bat into the obj depot directory.
- Edit wssetup.bat and change the line that modifies the classpath to point to the location of your various toolkit .jar files. You'll notice a remark at the top. You can cut and paste this line to your command line. When executed, this will add ObjDepot to the Apache SOAP deployed services. You can access the Apache SOAP Admin Web page at: http://localhost:8080/soap/admin/index.html. That way you can see that the Web service is properly deployed.

Removing from the Depository

There is also a remarked-out line at the bottom of the batch file that can be used to remove ObjDepot from the Apache SOAP depository.

- From a command window, change to the client directory and execute wssetup.bat. Compile the Java files in the client directory. Now go to the objdepot directory and compile the Java files there.
- Create a directory called "objser" under Tomcat's webapps directory.
- Move (not copy) ObjSerializer.class, ACla ss.class, and BClass.class to the objser directory.
- Modify your tomcat.bat file to include the objdepot directory in the classpath.
- Start Tomcat.
- From the client directory, run Client.

You should see indications that AClass was received and BClass executed.

Create Your Own

If you download the code, read the comments, and refer back to this article, you should be able to figure out how to use SOAP to transfer Java objects from your Web service and use them on a client.

Special .NET Focus Issue

Coming this February.
The most comprehensive .NET coverage!

WebServices
.NET J2EE XML **JOURNAL**

The definitive issue on **.NET**

SPECIAL: .NET FOCUS ISSUE
WebServices
.NET J2EE XML JOURNAL
WSJ2.COM
.NET: Wading through the Hype to the Facts

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

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

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

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

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

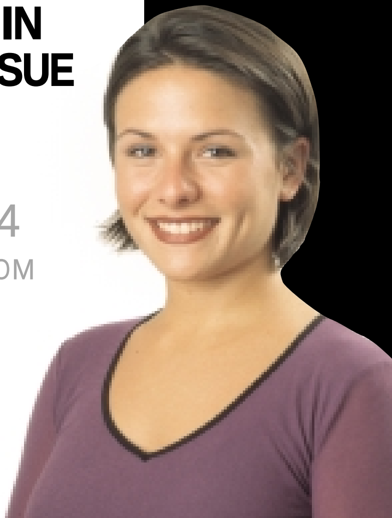
Microsoft Web Services Development Toolkit
A look at the new features available in this hot technology.
by Hitesh Seth

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

and more...

SUBSCRIBE ONLINE and SAVE
www.sys-con.com

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



SYS-CON MEDIA

Don't miss this special issue!

Written by Michael Sick

Web Services @ Work

Gluing Web services to Baan

Web services is often spoken of as a future technology, yet it's important to understand that Web services is already proving to be a key component of the products and projects of today.

In this article I examine how Web services has become an enabling piece of Epic Data's Connectware for Baan toolkit, which is designed to easily connect wireless devices and other clients to the Baan warehousing and manufacturing ERP modules. I then study the requirements and realities that led Epic Data to select Web services over competing technologies. Additionally, I examine the reasons why Epic selected The Mind Electric's GLUE as their Web services foundation. In summary, I hope to show that even though their role will grow in the future, Web services is at work for developers today.

Before I start, it's also worth noting that this is not meant to be a detailed technical study of a Web service architecture. There are no code samples, and concepts are discussed at a fairly high level. This seems appropriate as the article is intended to be an example of a Web service success and not an instructive how-to for building technology. Developers might use this and similar articles to convince technical managers to adopt or champion Web services within their organization. More technical questions can be forwarded directly to me.

Introduction to Epic Data

Epic Data is a products and consulting company that delivers solutions in the supply chain, wireless, warehousing, fulfillment, and ERP markets. Many of their solutions focus on wireless connectivity and data capture for large ERP systems such as Baan, SAP, and Oracle. These solutions have been used in over 2,000 installations worldwide and support many Fortune 1,000 businesses.

Introduction to Connectware for Baan

Connectware for Baan (CFB) is a well-established, industry-leading product that connects wireless devices (primarily bar-code readers) to the Baan ERP system. CFB is Epic's leading connectivity product and critical to the company's success. CFB's primary responsibilities are:

- Warehouse management
- Production reporting
- Label generation

An example and common use of CFB is the acceleration of receiving goods into a warehouse by the automatic assignment of serial numbers to incoming goods, the generation of appropriate labels, and the suggestion of efficient storage locations. The product acts as the middleware that ties together the barcode reader, label printer, and underlying ERP system. This process saves valuable time and reduces human error (see Figure 1).

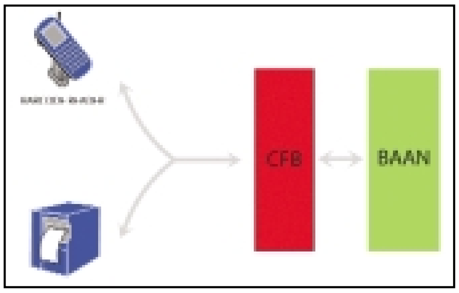


FIGURE 1 Overview of Connectware for Baan

Until recently, CFB was implemented as a set of Perl and C programs that managed all of the interactions between the client devices and Baan. This branch of the CFB solution has a broad install base of customers and a strong history of success. It's critical that any future versions of the CFB product line offer a smooth upgrade path.

In early 2000 Epic Data began work on a new version of the CFB product line designed around the following goals:

- ***Cross-platform services:*** The middleware solution would need to be easily ported/supported across the various flavors of Windows and Unix popular with Epic's customers.
- ***Cross-language clients:*** Because Epic data deploys these solutions in many heterogeneous environments, they're often charged with integrating the product into other systems written in various development languages. Epic wanted to support integration with popular development environments in a cost-effective and reliable manner.
- ***Cross-Baan support:*** Versions 4.X and 5.X of the Baan ERP suite are widely used and supported. This makes it necessary for Epic to support both versions of the product. Epic wanted a product which both insulated the client

> "It was the possibility of meaningful backwards compatibility that sealed the Web services/SOAP selection"

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 such positions as architect and VP of development.
MIKE_A_SICK@YAHOO.COM

developer from the differences found in these two versions but also provided access to version-specific features at a lower level.

- ***Standards-based:*** Epic Data decided to make a firm commitment to supporting open standards to reduce development, support, and integration costs and increase customer and partner prospects.
- ***Backwards compatibility:*** Because of Epic's significant customer/install base, it was essential that any new solution could seamlessly support the existing clients without disruption.

Eric Fritz, Epic Data's director of Baan product development, describes the need for greater product flexibility, saying:

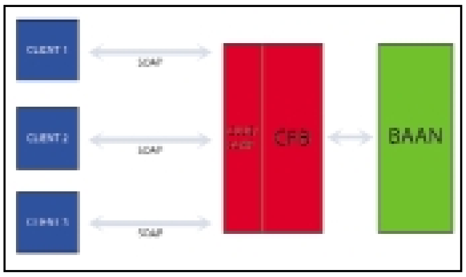


FIGURE 2 | **Architecture of Connectware for Baan**

"Our customers typically don't care what technology we use, but they do care about how much our solution may restrict them from making other business decisions. Yesterday if a customer told us they were going to move their servers from Unix to an AS400 we would have to undertake a major project to migrate their solution, and as we all know that means money out of their bottom line. Today, we can say 'No problem.' The same holds true for the version of Baan and the tools that they want us to use. It's all about the freedom to choose.

While the list of goals was much more extensive than what we have discussed, the above requirements are the ones that most impacted the product architecture.

The Selected Architecture

Epic Data made the decision that the core of the next generation Connectware for Baan product would be written in Java and that this functionality would be exposed as Web services. An architectural overview is shown in Figure 2.

Java was selected because it easily satisfied the cross-platform requirements

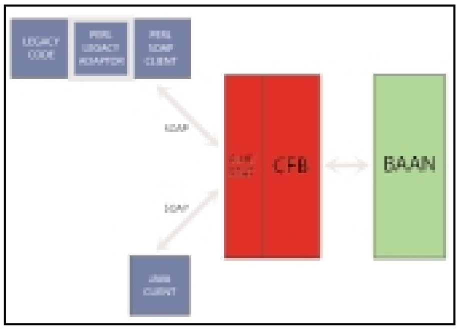


FIGURE 3 | **Legacy adaptor for Connectware for Baan**

and it also represented a significant commitment to open standards. Epic Data also leveraged Java's interface features and design by contract to expose cross-version features in a generic manner. The selection of Java, however, did not meet all of the above criteria.

To achieve integration with clients across multiple languages and facilitate backwards compatibility with legacy Perl and C code, Epic Data turned to Web services defined as WSDL, SOAP, and XML. CORBA was also considered as a possible solution but was determined to be overly complex and expensive to implement. SOAP was selected over other XML-based Web services because of the incredibly strong industry support that the standards are receiving in popular development languages like Visual Basic, Perl, C, C++, and Java.

Finally, it was the possibility of meaningful backwards compatibility that sealed the Web services/SOAP selection. Epic Data decided that they would create a Perl Legacy Adaptor to interface existing Connectware for Baan implementations with the new service. This Perl layer "intercepts" requests intended for the legacy back end and routes them to the new service via SOAP (see Figure 3).

GLUE: A Web Services Platform

Upon selecting Web services as a key part of their solution, Epic Data then had to select a Web services platform to enable their code. Ultimately, Epic selected The Mind Electric's GLUE. The Mind Electric describes GLUE :

"GLUE is 100% Java and has a rich set of features including an embedded Web server, servlet engine, SOAP processor, XML parser, graphical console, dynamic WSDL generator, dynamic Java/XML mapping, UDDI client, UDDI server, WAP support, and XML persistent storage system."

A recent review of GLUE in *Information Week* by Jason Levitt noted that GLUE's purpose was to bring the same simplicity seen in .NET to Java developers. GLUE succeeds by offering a high degree of Web services functionality combined with incredible ease of use. Epic Data selected GLUE for the following reasons:

- ***Relative maturity:*** At the time of selection, GLUE had a very competent implementation of SOAP 1.1 and great support for WSDL. GLUE also had an expressive implementation of UDDI's Publish and Inquiry functionality.
- ***Rapid advancement:*** TME's pace of development for GLUE would be rapid for a large company, let alone a typically understaffed startup. Releases are frequent and bugs are quickly reported on the product's mailing list and confirmed/fixed by TME staff.
- ***Ease of use:*** Both publishing and subscribing to Web services is made easy with GLUE. In both situations GLUE dynamically creates the necessary classes leaving the developer free from excessive hand-coding or messy code generation tools.
- ***Speed:*** GLUE was much faster than many of its competitors. TME has stated that this is mainly a result of using TME's proprietary XML parser and toolkit ElectricXML.
- ***Low vendor lock-in:*** The use of GLUE is nearly transparent on both the client and server sides. Epic Data estimates that less than 1% of the project code knows anything about GLUE or SOAP.

Jonathan Barksdale, software engineer at Epic Data, describes his experience with GLUE:

"I initially implemented and tested my would-be server methods locally. Then I added GLUE to the mix in a matter of minutes using only a few lines of additional source code. I can't believe the ease with which my formerly local application methods were working as distributed, standard Web services."

"By using Web services combined with Java on the server side, **Epic Data was able to build a solution** that was portable across languages, platforms, **and versions of Baan"**

While maturity, rapid product advancement, ease of use, and speed were all strong factors in its selection, the almost non-existent vendor lock-in made the final difference. GLUE, aptly named, binds cross-language ponents together and leaves a minimum of technical residue. For Epic Data to switch from GLUE to another Web services platform, there will be very little code investment to throw away. Essentially, the only thing Epic would need to do is the total amount of work required by the new solution. This all suggests a high degree of confidence by TME that GLUE will continue to be used on its own merits rather than maintained because it's painful and expensive to remove.

Summary

Epic Data, with a significant history of success in the Baan connectivity space, has made a significant commitment to Web services and to open standards. This investment was made not to satisfy a need for market hype but rather because these commitments offered tangible technical and cost benefits over alternative solutions. By using Web services combined with Java on the server side, Epic Data was able to build a solution that was portable across languages, platforms, and versions of Baan. Additionally, we saw Web services empower Epic to select a new development language and environment while preserving backwards compatibility.

Through the selection of GLUE, Epic ensured that Connectware for Baan's Web services were standards-compliant, high-performance, and on the leading edge of development in this new market. GLUE's lack of heavy coding requirements made it especially attractive, as Epic did not feel it was headed toward irreversible vendor lock- in. As CFB evolves, Web services are sure to remain a critical component of the overall architecture.

Web services is at work in a quiet but pervasive way across the computing industry. Epic's focus on these technologies is cutting-edge but certainly not unique. Web service-oriented architectures are being sketched on white boards around the world and have already made their way into a first wave of products and projects. While many of the underlying technologies are new, Web services already provide a better solution, in many cases, for cross-platform and cross-language computing than preceding technologies and standards. Web services is hard at work today and this trend shows every sign of increasing.

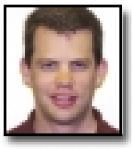
BEA eWorld

www.bea.com/events/world/2002/

Reviewed by Brian Barbash

SpiritWave from SpiritSoft

A flexible, reliable messaging system tool



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

SPIRITSOFT
100 Medway Road
Suite 203
Milford, MA 01757

TEST ENVIRONMENT:
Pentium III, 256Mb RAM, Windows 2000

PRICE:
$9,700 per developer
Server run-time: $5,500 per CPU

CLIENT RUN-TIME:
$5,000 per 50 users
$3,000 per adaptor per server
(Adaptors for IBM MQ series, RV5, RV6, ETX, MSMQ, etc.)

FOR SALES INFORMATION:
Tel: 1 508 473-3227
E-mail: sales@spirit-soft.com

EDITOR'S NOTE:
SpiritWave 5.0, the newest version of this messaging system, was released after this review was written. SpiritWave 5.0 is the first open JMS framework to provide multi-plug adaptors to integrate any proprietary legacy message-oriented middleware (MOM) system. These multi-plug drivers enable easier development of distributed systems by neutralizing the dependency on vendor-specific interfaces and by managing these proprietary MOM systems so IT managers don't have to. SpiritWave 5.0 includes drivers for all leading proprietary MOM products such as IBM MQSeries, MSMQ, Tibco Rendezvous, and Tibco ETX.

SpiritWave 4.4 from SpiritSoft is a vendor-independent, JMS-compliant messaging system that provides a common interface to heterogeneous messaging systems. It supports the publish/subscribe and point-to-point messaging paradigms. To leverage existing investments in messaging systems, SpiritWave provides transport drivers for IBM's MQ Series, Tibco's Rendezvous, Microsoft's MSMQ, OpenTrade's Orbita, and Active Software's ActiveWorks products. It also provides a pure Java JMS implementation for both queue-based and publish/subscribe messaging. Figure 1, from SpiritSoft's Web site, illustrates the overall architecture in which the SpiritWave messaging system interacts with third party transports.

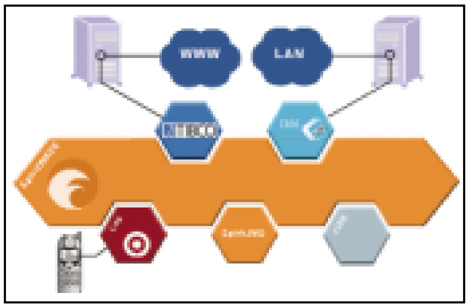


FIGURE 1 SpiritWave architecture

Working with SpiritWave

Installation and Administration

Installation of SpiritWave is straightforward and is available for both Windows and Unix systems. For this review, I've installed SpiritWave on a Windows 2000 Professional machine, with a Pentium III and 256 Mb of RAM. Once installed, the administration and message queue applications are available along with several sample applications that supplement the documentation.

SpiritSoft provides a common administration architecture for all of its products, called SpiritAdmin. Configuration information is housed in XML data files that may be stored in various formats, including a JDBC-compliant database.

SpiritWave's provided messaging transport comes in two flavors: Standard and Enterprise. The two drivers provide the same services with regards to message transportation, but the Enterprise component adds the capability to validate users and control access to destinations. It's optimized for a distributed enterprise messaging, whether queueing or publish/subscribe. Each messaging transport is fully configurable from the SpiritAdmin console provided.

As mentioned earlier, SpiritWave contains several modules to allow message transport systems to become accessible via the JMS API. For the modules that provide access to the provided message queue, three options or modes exist: Unadministered, Administered, and Enterprise. Under the Unadministered configuration, the messaging system and its communicating clients aren't controlled from a centralized configuration mechanism. Configuration details are set up by the individual application and may be stored independently, often within a JNDI tree. In the Administered mode, the configuration details for the transport module are stored within the repository established for the SpiritAdmin framework. This repository may be, for example, XML files or a JDBC database. In the third option, Enterprise, the modules incorporate the user management and destination access services components of the SpiritAdmin architecture.

Development

For this review, I've created a simple application that can accept XML requests from various external sources through a messaging subsystem. The messaging subsystem will be implemented with SpiritJMQ, a JMS-compliant message transport. I've placed the connection and queue into a JNDI tree inside IBM's WebSphere 3.5.3 application server. SpiritWave also provides integration with BEA's WebLogic application server through predefined registration classes and deployment descriptors.

The first task in working with SpiritWave is to establish and bind the queue connection factory and a default queue into the JNDI tree. To accomplish this, a simple registration servlet executed at server startup will create the necessary objects and perform the binding. Once in the JNDI tree, the objects are available to clients and message-driven EJBs for processing.

If the developer is working with WebLogic, the registration process is straightforward and easy to configure. SpiritSoft provides a startup class that takes no parameters and reads its configuration information from an XML file. The configuration file specifies all communication properties for SpiritWave and the JNDI binding information.

Connections to the SpiritWave messaging system may be obtained in two ways. The first uses the standard JMS connection factories, queue or topic, appropriate for the task at hand. The second incorporates SpiritWave's WaveProfile object that defines the implementation details of the underlying transport. If a developer chooses the WaveProfile approach, it's incorporated in the creation of the connection factory.

WaveProfiles separate the message transportation and storage mechanisms to provide flexibility to the developer and the application environment. For storage, the developer may choose between the object-oriented databases from ODI and Versant, and a standard relational database accessed via JDBC. For this review, I've chosen to use a JDBC connection to Oracle 8*i*.

After retrieving a JMS connection, a session is created through the standard procedure. The developer must specify the acknowledgement option and whether or not the session is transacted. When using transacted connections, no messages are delivered until specific commits are issued from the publishing object. Subsequently, messages aren't acknowledged until specific commits are issued from the message consumers.

JMS destinations may be obtained using either the standard JMS approach or the proprietary option available through SpiritWave. If a developer is using JNDI, as in this review, the standard method must be used to obtain a topic or queue. If, however, an alternative repository contains the destination object, the developer has the choice to use lookup methods on the SpiritWave class appropriate for the object desired. All objects are identified by a logical name. Future releases of SpiritWave will support JNDI.

For the remaining process of transferring messages through the system, creating the message, sending, receiving, and managing the connections, SpiritWave relies on the standard JMS API calls.

SpiritWave provides several enhancements to the messaging system that provides further control and management of items. They include:

- ***Subscription Listening:*** Message publishers can determine if topics or queues have been subscribed to prior to publishing a message.
- ***Connection Inbox:*** When a subscriber connects to a topic, the raised subscription event's reply To method returns the inbox on the current connection. This may be used to send a unique message to the subscribing client.
- ***Wildcards:*** For topic subscribers, wildcards may be used to identify the topics the application should subscribe to.
- ***Retrieve the Last Message:*** SpiritWave's JMQ driver provides the capability for subscribers to retrieve the last message sent to a topic without subscribing. This can facilitate the initialization processes for the subscribing agent.

As mentioned earlier, SpiritWave provides several sample applications to provide guidance for new developers. These examples are very helpful and cover a full range of development scenarios, including queue-based messaging, publish/ subscribe messaging, working with JMS and JNDI, development with the Enterprise version of JMQ, and examples for some of the enhancements to the API listed above. Several of these examples provided guidance as I set up my application for this review.

Summary

SpiritSoft's SpiritWave provides a tool that allows existing messaging systems to be leveraged and extended by the JMS API. With its ties to the major messaging vendor products such as Tibco Rendezvous and the IBM MQSeries, along with its supplied messaging system, SpiritWave presents a flexible and reliable framework on which to build message-oriented applications.

.NET FEATURE

written by Derek Ferguson

Invoking .NET Web Services from Mobile Devices Part 1 of 2

Pocket PCs put .NET Web services at your fingertips

In May 2000 I was invited to Microsoft's corporate headquarters in Redmond for a special "technology experts" summit. At this summit, the forty or so of us in attendance were given a special sneak preview of a technology upon which Microsoft planned to "bet the farm," so to speak. They called the technology "ASP+ Web Methods."

Of course, nowadays we all know these as .NET Web services. The fundamental technologies upon which they are based (XML, SOAP, SDL, UDDI, DISCO, etc.) are probably familiar to just about all of this publication's readership in one form or another.

What many Web service developers are not aware of, however, is how good .NET is at "playing nice" with all sorts of different mobile devices. The Mobile Internet Toolkit, for example, adds out-of-the-box support for WAP and i-Mode clients to .NET's already formidable Web application infrastructure. Similarly, the Smart Device Extensions for .NET and the .NET Compact Frameworks bring the power of the .NET Common Language Runtime (CLR) environment to a wide range of wireless Internet devices, such as the Pocket PC.

The focus of this magazine, however, is on Web services. For this reason, in this article I'll show you how .NET Web services may be accessed and utilized from many different devices.

The code excerpts in this article have all been taken from my book, *Mobile .NET* (Apress, 2001). If you find this article interesting, I recommend that you purchase a copy!

Creating the .NET Web Service

When it comes to teaching people Web services, stock-quoting applications seem to have taken over the role that "Hello World" served so elegantly with older technologies. This is to say that a Web service capable of returning information – or simulated information – about the current price(s) of various stocks seems to be the example of choice in better than 90% of all Web service literature out there.

So, for our example mobile Web service, we'll develop a Web service that takes a single, four-letter stock symbol as its sole parameter and returns a single string to indicate the current price of this stock. To fuel the Web service with up-to-the-minute data, we'll make use of Lycos' excellent Finance Web site.

To create the sample Web service, follow these steps:

1. Open the Windows Start menu.
2. Locate the Microsoft Visual Studio.NET 7.0 Program Group.
3. Launch Microsoft Visual Studio.NET 7.0.
4. Click "New Project" on the start page.
5. Select a Project Type of "Visual Basic Projects".
6. For a template, choose "ASP.NET Web Service".
7. Name your Web service "Chapter10Serv er" and click "OK".
8. In the Solution Explorer, right-click "Service1.asmx".
9. Choose "View Code" from the pop-up context menu.
10. Enter the code shown in Listing 1.

What does it all mean?

Now that you've entered all of the code for your Web service, it makes sense to pause for

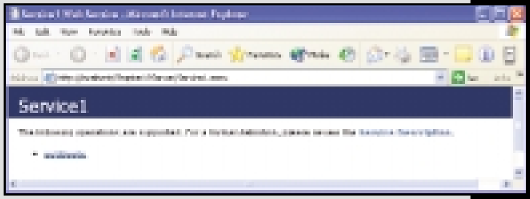

Service1

FIGURE 1 Web service front end with intranets/databases behind

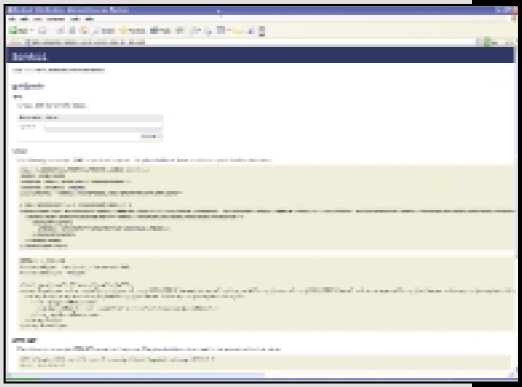
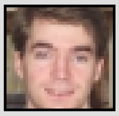


FIGURE 2 Web service front end with intranets/databases behind



Author Bio:
Derek Ferguson is chief technology evangelist for Expand Beyond Corp. He is a world-renowned author, speaker, and developer who has been recognized throughout the IT industry for his work.

DEREK@XB.COM

JavaOne

www.http://java.sun.com/javaone/

a moment to review exactly what it all means.

The first two "imports" lines in Listing 1 tell .NET to include support for Web services within our Visual Basic application. Specifically, the first line tells .NET to include general support for Web services. The second line tells .NET to include support for Web service protocols other than .NET's native Web services protocol.

This second line is very important. Looking at the industry as a whole, simple SOAP is still the most common protocol used for invoking Web services. Many of the toolkits out there for building Web services clients do not yet support Microsoft's full Web services protocol. For this reason, it's easier to make .NET "speak their language" than vice versa!

After these first two lines, we have a couple of lines that ask .NET to include support for low-level networking (e.g., TCP/IP socket communications) and regular expression parsing. Regular expressions are a new addition to Visual Basic with .NET, and will be greatly welcomed by those with previous experience in other languages such as Perl.

We make our class into a Web service under .NET by inheriting it from System.Web.Services.WebService. The tag preceding our class declaration is .NET nomenclature for declaring a namespace for our Web service.

A similar tag is prepended to our getQuote method's declaration to indicate that this method should, indeed, be exposed to the Internet as a Web service. However, within this tag there is also the indication "SoapRpc Method()." This is how we leverage .NET's support for industry-standard Web services protocols to expose our method as a standard remote SOAP method.

Our getQuote method takes a single parameter, the symbol of the stock for which we are interested, in receiving a quote. The first thing our method does is use .NET's regular expressions parsing capabilities to verify that the symbol consists of exactly four letters. If this isn't the case, an exception is thrown and the Web service call terminates.

Assuming that the symbol is properly-formed, however, the next thing that our method does is to pass this symbol along to Lycos' Quote.com Web site to get a price quote. This is accomplished by using .NET's TCP/IP capabilities to send a simple Web request across the Internet. The contents of the page returned are stored as a (very long) string is the strResponse variable.

At this point, we once again leverage .NET's regular expressions to parse the Web page for the first occurrence of the phrase "Last Sale" that is followed by numbers, a decimal, and then more numbers. This is the general area of the page where our most recent stock price has been returned.

From within this section, the numerical portion is finally parsed out. This is the value that is returned by our method as the results of any Web service invocation.

Creating the Pocket PC Client

In many ways, Pocket PCs are probably Microsoft's favorite mobile devices. For this reason, we'll begin by showing you how to invoke .NET Web services from these devices. To follow along with this discussion, you'll need:

1. Microsoft eMbedded Visual Basic, and
2 Either a Pocket PC, or a Pocket PC emulator
3. Pocket SOAP

eMbedded Visual Basic and a Pocket PC emulator can both be downloaded for free as part of Microsoft's eMbedded Visual Tools. The URL to this download site is: www.microsoft.com/mobile/downloads/emvt30.asp.

The package will come to you as a standard Windows installer. Simply walk through the installation wizard and make sure that you choose to install eMbedded Visual Basic and (if you don't have a "real" Pocket PC) the Pocket PC Software Development Kit (SDK).

Simon Fell's Pocket SOAP software is currently the most popular tool for invoking SOAP-style Web services from Pocket PCs. To obtain this and install it onto your device or emulator, follow these steps:

1. Browse to www.pocketsoap.com.
2. Follow the links to version 1.1 of the software.
3. If you have a "real" Pocket PC, download the Pocket PC install. If you have the Pocket PC emulator, download the x86 binaries.

The Pocket PC Install

Putting the Pocket PC install onto an actual Pocket PC device is a fairly easy process. First, make sure that your PC is connected to your computer and that ActiveSync is functioning properly. Then, run the Pocket PC install. It will walk you through an installation wizard, then ActiveSync will finish the installation of the software onto your device.

Installing the x86 binaries onto the emulator is a little more complicated. You must:

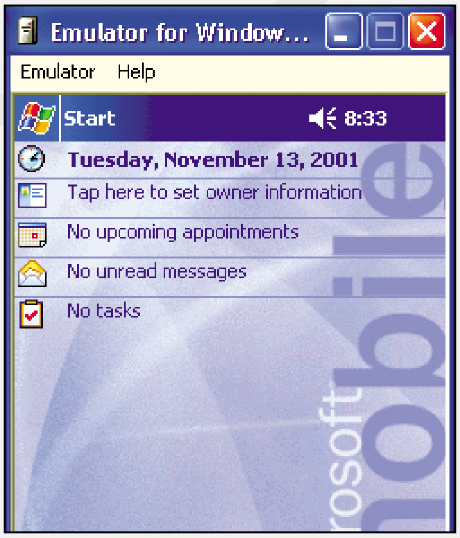
1. Unzip the archive that you downloaded.
2. Open the Windows Start menu.
3. Locate the Microsoft Windows Platform SDK for Pocket PC program group.
4. Start the "Emulation Environment for Pocket PC" within this group.
5. You are now presented with a command prompt at a certain location.
6. Copy the "pSOAP.dll" from the archive in step #1 above to the location in step #5 above.
7. Within the Emulation Environment, type "regsvrce".
8. Once the Pocket PC emulator starts, type "\windows\pSOAP.dll" where asked for the full pathname, then click "OK".

You are now (finally) ready to create the code for the Pocket PC Web services client. Simply choose to create a new Pocket PC project under eMbedded Visual Basic, add a single text box and command button to the project's default form, and then add the code from Listing 2 to the command button's Click event.

The first several lines of code in Listing 2 are concerned with creating two of the most important objects in Pocket SOAP: the Envelope and the Transport. A Pocket SOAP Envelope is where the body of the SOAP message is assembled.

The Transport is responsible for making sure that the message created by the Envelope gets from the device to the Web service, and for receiving the Web service's response. Several kinds of Transport are possible within the framework of SOAP: SMTP, FTP, etc. For our purposes, we've chosen to use the HTTP Transport, which will convey our messages using standard Web traffic.

The next three lines of Listing 2 set various properties for our SOAP request. Specifically, the URI must be set equal to our Web service's namespace. The MethodName should be the name of the method within our Web service



that we wish to invoke. Finally, you should call CreateParameter with the name and value of every parameter that you must pass in.

The next thing we do is call the Serialize method on our Envelope object to translate our message into a simple string. We store this string in a variable and then call MsgBox to display it for your inspection. This is strictly for demonstration and debugging purposes. In a true, production Mobile .NET application, this step would typically not be taken.

We call the Send method on our HTTP Transport object and pass in the URL of our Web service and our Envelope as parameters. This is how our SOAP request travels from the Pocket PC to the remote Web service.

We then call the Receive method on our HTTPTransport object – this call blocks until our Web service responds. The response is returned as a string which we store in a variable named strResponse.

By passing this string into our Envelope object's parse method, we're able to translate the text message that our Web service sent back into a real SOAP response. The code concludes by extracting the price of our stock out of this response and displaying it in a message box for the user.

In Conclusion

As you've seen in this article, creating a .NET Web service that can be accessed and utilized from a mobile Internet device is easy! Thankfully, Simon Fell has provided an excellent toolkit in his Pocket SOAP software for invoking .NET Web services from Pocket PCs.

Recently, however, Microsoft publicly released a brand new technology that promises to make calling .NET Web services from Pocket PCs even easier. This technology is known as the .NET Compact Framework and will be discussed at length in the next installment of this two-part series.

In the meantime, if you have any questions or comments, please contact me via my Web site at www.MobileDotNet.com.

Listing 1

```
Imports System.Web.Services
Imports System.Web.Services.Protocols

Imports System.NET
Imports System.Text.RegularExpressions

<WebService(Namespace:="http://xb.com/webservices/")> Public
Class Service1
    Inherits System.Web.Services.WebService

    <WebMethod(),SoapRpcMethod()> Public Function
getQuote(ByVal symbol As String) As String

        Dim hwreq As HttpWebRequest
        Dim hwres As HttpWebResponse
        Dim inStream As System.IO.Stream
        Dim strResponse As String
        Dim rgExp As Regex
        Dim m As Match

        m = rgExp.Match(symbol, "[a-zA-Z]{4}")

        If m.Success Then

            hwreq =
CType(WebRequest.Create("http://finance.lycos.com/home/stock
s/quotes.asp?symbols=" & symbol), HttpWebRequest)
            hwres = CType(hwreq.GetResponse(),
HttpWebResponse)
            inStream = hwres.GetResponseStream()

            Try
                Do While True
                    strResponse = strResponse &
Chr(inStream.ReadByte())
                Loop
            Catch ex As Exception
            End Try

            inStream.Close()

            Dim temp As String

            m = rgExp.Match(strResponse, "(?:Last
Sale.*)\d+\.\d+", RegexOptions.Singleline)

            If m.Success Then

                temp = m.Value()

                m = rgExp.Match(temp, "\d+\.\d+",
RegexOptions.Singleline)

                If m.Success Then
                    getQuote = m.Value()
                    Exit Function

                End If

            End If

        End If

        throw new Exception("Invalid stock symbol!")

    End Function

End Class
```

Listing 2

```

Private Sub Command1_Click()

    Dim strRequest As String, strResponse As String
    Dim psEnvelope As PocketSOAP.CoEnvelope
    Dim psTransport As PocketSOAP.IHTTPTransportAdv

    Set psEnvelope = CreateObject("PocketSOAP.Envelope")
    Set psTransport =
CreateObject("PocketSOAP.HTTPTransport")

    psEnvelope.MethodName = "getQuote"
    psEnvelope.URI = "http://xb.com/webservices/"
    psEnvelope.CreateParameter "symbol", Text1.Text
    strRequest = psEnvelope.Serialize
    MsgBox strRequest

    psTransport.SOAPAction =
"http://xb.com/webservices/getQuote"
    psTransport.Timeout = 99999

    psTransport.Send
"http://localhost/Chapter10Server/Service1.asmx", strRequest
    strResponse = psTransport.Receive

    psEnvelope.parse (strResponse)
    MsgBox psEnvelope.Parameters.Item(0).Value

End Suba
```

Download the code at
sys-con.com/webservices

It's About More Than Just the Plumbing

The real issues that need to be solved are the nontechnical ones

WRITTEN BY

Simon Phipps

Simon Phipps, currently chief technology evangelist at Sun Microsystems, speaks frequently at industry conferences on the subject of technology trends and futures. He was previously involved in OSI standards in the 1980s, in the earliest collaborative conferencing software in the early 1990s, and in introducing Java and XML to IBM. This article represents his personal opinions and not necessarily those of any employer. Simon can be contacted via www.webmink.net.

SIMON@WEBMINK.NET

I've described elsewhere the idea of "swarms" – spontaneously federating devices and software services connecting over networks. Some people are now describing this concept as "wireless Web services," extending the group of ideas now being called *services-on-demand*.

As usual, the computer industry is keen to address the details of protocols and connections, but is leaving until later the real, people-centric issues that technologies can't solve.

Possibilities

Imagine the possibilities once we can have the services not only offered to us on demand via the nearest device but also have them include both our current context and our online profile. I'll be able to tell my car who to start for and how each person may use it ("yes, you can borrow it, but don't drive home if you're drunk, and stay under 50"). As my current meeting ends late, my PDA will offer suggestions for how to reschedule the rest of the day's meetings and travel. The just-in-time production line will be able to respond to all the inputs, including current demand in the retail stores. All sorts of ideas that we thought were the domain of intelligent agents will start to become a reality.

But hold on a moment. Can it all really happen like that? For some time now my thesis has been that we already have, or can imagine, all the technology we need to build these spontaneously federating solutions. The real issues lie elsewhere.

Standards

The first priority has to be open, loosely coupled standards, for content as well as infrastructure. In the area of standards, it's clear that we've made progress over the last decade. The Web browser gives me a standard space to interact with remote computers. GSM gives me a mobile phone that works worldwide. But we're not there yet with Web services, let alone with *wireless* Web services – none of the basic technologies is actually an open, royalty-free standard today. Discussion over the actual XML content of the transactions, although in progress at ebXML, is in its youth and under-supported by the key vendors. And peer-to-peer ideas tend to be neglected altogether.

If we've learned one thing from a decade of the Web, it's that the massively connected mesh demands open, loosely coupled standards – "open" in the sense of available to anyone to develop with or use without fee; "loosely coupled" in the sense of tolerating extension without opening a path to proprietary lock-in; "standards" in the sense that a democratic process offers every affected developer and user the chance to be involved in each change.

For services-on-demand (both Web services and P2P services) – to be offered to the mobile user, we'll need a serious commitment to openness and standards that extends beyond the mode of merely reconciling infrastructure for proprietary content that we're seeing at the moment from some.

Business Models

Secondly, we'll need some careful progress made on business models. Web services void the only business model that "free" services on the Web ever had – that of exploiting eyeballs. Without advertisements, many information providers will have to look elsewhere for a revenue model. So far as European telcos are concerned, the urgent need to validate their investments in 3G licenses may lead to progress here. We may see commercial-quality service federations created that are free at your desk but part of the service plan on the move – as long as we can overcome the contractual, legal, and intellectual property problems.

Logistics

The procedural barriers may, thirdly, prove to be the killers. Realistically, no one vendor or service provider will ever be able to provide everything you need – it will take a community. Building communities is the lifeblood of progress on the Web and has underpinned Web standards (W3C is essentially an expert community), open source development (from pioneers like Linux and Apache to commercial foundations like NetBeans and OpenOffice), and technology evolution. But the application space gets more complex once we go mobile. What about antitrust laws? What about intellectual property ownership? What about contractual protection?

And then there are the international issues – local languages, differences in jurisdiction, and so on. Can my swarm continue to work across state lines? How about across international boundaries? Or with local services in a country with a different local language?

Opportunity

After reading this far, you might think I'm pessimistic about services-on-demand being offered to the mobile user. But, on the contrary, I'm very optimistic. In the mobile space, in the absence of the monopolistic pressures that have impacted the PC market, we've seen extensive industry agreement. Up to this point, we've seen agreement on GSM and Java technology deliver a level playing field to the network operator and to the developer.

The plumbing is in place. But people and business issues have always been the linchpins in making new technologies real in the mass market. It's true for wireless Web services-on-demand as well.

Wireless Web Services with J2ME

Remote possibilities

What happens when the hype of Web services meets the increasingly popular and ever-changing world of wireless computing? Most likely, confusion and disillusionment. In this two-part article, we'll explore the uncharted waters of wireless Web services. We'll use the J2ME platform for developing our Web service clients and access remote services on the Internet using standardized industry protocols. In this first article, we'll examine XML-RPC, a simple, lightweight mechanism for invoking remote services with XML. The second article will compare and contrast XML-RPC with SOAP, a more robust, sophisticated, and heavier solution for invoking remote services with XML.

The Wireless World

A Web service is a coarse-grained interface to one or more business services that is invocable across a network. With a wireless network, this invocation process becomes more complicated. Many cellular telephone service providers use analog circuit-switched networks that open a constant connection for the duration of the exchange. More advanced providers are moving to digital packet-switched networks. In packet switching, a stream of digital bits is carved up into bit clusters, called packets, and blasted across the network individually.

Circuit-switched analog networks are more expensive to maintain and offer limited bandwidth. Digital packet-switched networks are cheaper, more efficient, and do not have the same bandwidth limitations. The trade-off with packet-switched networks is that packets are occasionally lost ("dropped") during transmission. Dropped packets must be retransmitted. The larger a transmission is, the greater the likelihood that packets will be dropped, requiring retransmission and degrading performance.

The bottom line is that regardless of the type of network being used by a provider, wireless clients must keep their exchanges as thin as possible to ensure optimum performance. Additionally, mobile devices typically do not have an abundance of resources for processing fat requests or responses, or storing robust data models.

XML-RPC for Wireless Web Services

XML-RPC is a Remote Procedure Calling protocol that invokes remote procedures over a network by sending XML-formatted messages. The XML-RPC specification was developed and is maintained by UserLand Software, Inc.; the full specification can be found at www.xmlrpc.org/spec.

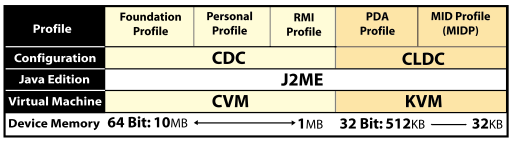
XML-RPC is an extremely lightweight mechanism that can be used as part of a Web services architecture. The key to a Web services architecture is the utilization of XML as a language-agnostic, vendor- and platform-neutral medium for accessing Internet or intranet services. XML-RPC provides the minimum functionality necessary to specify data types, pass parameters, and invoke remote procedures in a neutral way.

What makes XML-RPC so efficient? XML-RPC defines eight data types: six primitive types (int, Boolean, string, double, datetime, and base64) and two complex types (struct and array). These are the only types available, yet they provide all the functionality that is needed about 80% of the time. Although SOAP provides a more robust data-typing mechanism based upon XML Schemas (even allowing the creation of custom data types), this is often overkill in a wireless environment. We'll explore these topics further in our next article; for now, we simply need to understand that XML-RPC is an extremely lightweight mechanism for invoking Web services in a standardized and neutral manner.

The wireless applications that we'll be developing in these articles require the J2ME platform, so we'll take a brief look at J2ME to provide for a basic background for these wireless Web services.

J2ME Primer

The Java 2 Micro Edition is a Java 2 platform for developing applications for devices with limited memory. Specifically, J2ME addresses the need for application development for consumer and embedded devices. Because it is designed for devices with extremely small footprints, many of the features of the J2SE are not included. Some of the notable features not included are floating point data types, serialization (no JavaBeans), thread groups and thread daemons, finalizations, user-defined class loaders, and the JNI. As Figure 1 indicates, the J2ME platform is a layered stack consisting of a virtual machine and the core J2ME class libraries, as well as configuration class libraries and device-specific profiles.



Profile	Foundation Profile	Personal Profile	RMI Profile	PDA Profile	MID Profile (MIDP)
Configuration	CDC			CLDC	
Java Edition	J2ME				
Virtual Machine	CVM			KVM	
Device Memory	64 Bit: 10MB ⟷ 1MB			32 Bit: 512KB —— 32KB	

FIGURE 1 The J2ME stack

Configurations

Configurations define the run-time environment by specifying the Java features (classes) that are available as well as which virtual machine will be used. A configuration can also be thought of as relating to a category of devices that have common characteristics and me-

Author Bio

Kyle Gabhart is the director of the Java Division of Objective Solutions (www.objectsoln.com), a high-end engineering services company based in Richardson, TX. Kyle is a prolific writer, with more than a dozen technical articles and books to his name. He is the original contributor of the kXML-RPC source code, which is now part of the EhydraME platform.
JAVA@OBJECTSOLN.COM.

Jason Gordon is an associate architect for Verizon's Technology Integration and eInfrastructure group (www.verizon.com). He is a core member of the kXML-RPC design team for EnhydraME and also serves as Region 5 Telecommunications Chair for the National Society of Black Engineers (www.nsbe.org).
JASON.GORDON@VERIZON.COM

“More often than not, XML-RPC will provide you with all the functionality that you need”

mory constraints. For devices that have a total memory from 160 to 512 kB, the Connected Limited Device Configuration (CLDC) for J2ME can be used. CLDC devices usually include cell phones, two-way pagers and low-end PDAs. The CLDC also targets devices with a network connection and processing power of 16 or 32 bits. The CLDC uses the K (k for kilobyte) Virtual Machine or KVM. For devices that have a total memory of 2MB or greater and a 32-bit or 64-bit microprocessor, the Connected Device Configuration (CDC) is used. The CDC uses the CVM and is generally used on set-top TVs, higher-end PDAs, and next generation mobile devices. A configuration (and corresponding virtual machine), combined with a device-specific profile, and the core J2ME libraries, constitutes a complete J2ME environment.

Profiles Overview

Profiles work on top of configurations and focus on a “vertical” market or industry segment of devices. Profiles also allow developers to address more device-specific features such as the life cycle of an application, user interfaces, and networking issues. CDC devices typically use the Foundation profile, which targets devices that require more networking capabilities and no GUIs. CLDC devices typically use the Mobile Information Device Profile (MIDP). For the wireless development that we focus on in this series, we’ll be using MIDP.

MIDP

The MIDP consists of APIs for user interface design as well as for database activity. A MIDP application is referred to as a *midlet*. MIDP even allows multiple midlets to be packaged together as a midlet suite and share information between midlets within the suite. This is generally only useful, however, in the case of midlets that maintain a database. For our purposes, we’re interested in MIDP’s GUI capabilities. MIDP supports 10 GUI components: Command, Alert, Choice, Choice Group, Form, List, StringItem, TextBox, TextField, and Ticker.

In our sample application, we’ll be using the following GUI components:

- ***List:*** Contains a list of choices, typically relies upon a device’s “select” or “go” functionality.
- ***Command:*** Presents a choice of action. Contains a label, a type, and a priority.
- ***Display:*** The midlet’s canvas upon which UI components are displayed.
- ***Alert:*** Informs the user about an exceptional condition or error. It can also be used to display the results of a query to the user.

To understand how a MIDP user interface is created and how it functions, it’s necessary to understand the life cycle of a midlet. This can be seen in four stages, each with a corresponding method defined within the midlet:

- ***Initialization:*** *constructor*: Every midlet has a default constructor. This is used to initialize a midlet’s data members, including GUI components, with their desired property values (size, shape, color, label, text, reference, etc.).
- ***Activation:*** *startApp()*: Acquires necessary resources, makes display visible to user, and begins to perform requested services.
- ***Passivation:*** *pauseApp()*: Stops performing services and releases shared resources.
- ***Destruction:*** *destroyApp()*: Releases shared and local resources and saves any persistent data.

The mobile device will handle the management of a midlet through these life cycle methods via Application Management Software (AMS). AMS frees the developer from directly managing a midlet and its resources.

Writing a MIDP Web Service Client

In this article, we’ll create a MIDP client that uses the XML-RPC protocol for invoking remote Web services in a platform- and language-neutral way. To do this, we’ll need a J2ME implementation of the XML-RPC protocol. At the time of this writing, the only publicly available client implementation is kXML-RPC, an open-source XML-RPC project for the J2ME platform. kXML-RPC is maintained by the Enhydra organization and can be freely downloaded from their Web site at http://kxmlrpc.enhydra.org. The kXML-RPC library uses Enhydra’s kXML parser to handle the low-level XML parsing details. With the addition of the parser, the kxmlrpc jar file reaches a whopping 24kb!

With the kxmlrpc JAR file downloaded to your system and placed in your application classpath, you can write the MIDP client. We’ll walk through the creation of the midlet and highlight the most interesting lines of code, but the entire source code for the midlet can be seen in Listing 1, and the source for MyMidlet.java can be downloaded from the kXML-RPC Web site (kxmlrpc-samples.zip) located at http://kxmlrpc.enhydra.org/software/downloads/index.html.

The first step, obviously, is to import the necessary packages and declare the MIDP components that will be used in the application. After this, we define the midlet’s constructor, initializing all the UI components and adding them to the display as necessary. With that complete, we need to fill in the three other life cycle methods. In the startApp() method, we simply bring the MIDP display into action. Since we don’t use any shared resources, the pauseApp() method is blank. Finally, the destroyApp() method releases the local resources that we have allocated for our midlet.

Now we’re ready for the interesting part of the code, the commandAction() method. This method is called anytime the user performs a command event (pressing a key, selecting an

“Wireless clients must keep their exchanges as thin as possible to ensure optimum performance”

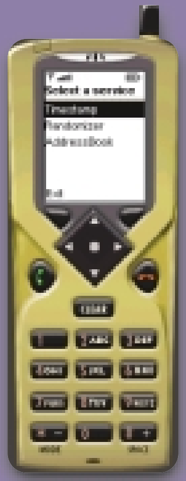


FIGURE 2 | XML-RPC demo midlet

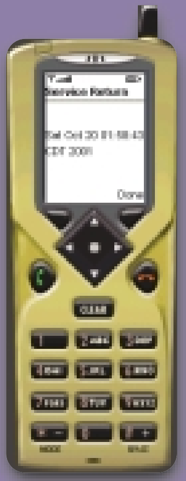


FIGURE 3 | The Timestamp service

item from a list, etc.). The Command and Displayable objects are then queried to determine which component has actually been activated/deactivated, and the appropriate actions are performed. Our midlet has three remote XML-RPC Web services displayed in a list (see Figure 2), and a switch statement is performed on the index of that list to determine which item has been selected. In Listing 1, only the first service is given an implementation, but the other two can be seen by downloading the source code.

The Timestamp service is very simple, a parameterless request is sent to the service and a String object representing the current time is returned. To perform this query, a kxmlrpc object is created with the specified URL for the Web service. Then an empty Vector is created and the actual request is performed with the following line:

```
String serverTime = ( String )
xmlrpc.execute(
"sysTime.getSystemTime", params );
```

The execute() method accepts two parameters, a String representing the name of the service, and a Vector representing any parameters that should be passed to the service. This particular service returns a String object that is then sent to the screen to display the current time on the server. An example can be seen in Figure 3.

Deploying and Testing a Midlet

For deploying and testing our midlet, we used Sun's J2ME Wireless Toolkit (J2MEWTK) version 1.0.3 beta which can be downloaded from Sun's Web site at http://java.sun.com/products/j2mewtoolkit/. The toolkit is 100% Java, built using the Java 2 Standard Edition, so even though it contains the J2ME APIs and is used for deploying and testing J2ME applications, it requires a J2SE implementation in order to run.

To deploy and test your midlet, you need to do four things: create a project for your application, write the midlet's code, place all of the files and resources in their appropriate application directories, and then build and run the application.

With the toolkit properly installed, the first step is to create a project for your application. To do so, follow these steps:

- Start the KToolbar application
- Click the "New Project" button. Name your project and name your project's midlet (this will also be the name used for the midlet in the .java source file).
- Click the "OK" button on the Settings screen that shows keys and values. This screen represents your application's deployment properties. You can specify these properties now or later by clicking the "Settings" button.

Now that you've created a project, the toolkit has created a corresponding directory structure for your project. That project directory structure is located under the apps directory of the J2MEWTK installation directory. We're only interested in three of them: the source code, resource, and library directories.

- \src: Place your java midlet's source code files in this directory.
- \res: Place any resource files (images, text files, etc.) in this directory.
- \lib: Place JAR files and Java class files that your midlet(s) will need in this directory.

After creating a project, writing the midlet code, and placing all the necessary files in the appropriate directories, you're ready to actually test the application. This requires three essential steps:

- Build the application into an executable midlet by clicking the "Build" button.
- Resolve any errors or exceptions that are thrown and rebuild the application until a successful build is accomplished.
- Execute the application by clicking the "Run" button after a successful build has been created.

When you run a midlet from the J2MEWTK, a phone emulator is started and the toolkit attempts to load your midlet into the emulator. You can test your midlet with any of the supplied emulators, or even download additional emulation environments from the Web. With your midlet running, you can navigate through your midlet just as you would on a real J2ME-enabled wireless phone. If your computer is currently connected to the Web, then you should be able to access the services listed in your midlet code.

Deploying into Production

Once your midlet development and testing is complete, you can package your application into an executable format by selecting the "Package" menu item from the "Project" menu. The toolkit will create a .jar and .jad file in your project's \bin directory. The .jad file is used for describing and executing your midlet, while the .jar file contains the Java class files, library and resource files used by your midlet. From now on, simply double-clicking the .jad file will run the midlet.

Looking Ahead

In this article we've taken a look into the world of the wireless Web, XML-RPC as a Web service communication protocol, and the J2ME environment with special attention paid to the MID profile, and also looked at a demonstration of an XML-RPC midlet using the kXMLRPC code. XML-RPC provides a very thin, efficient means of invoking remote services in a standard and neutral way. It defines a succinct set of eight data types, providing the means necessary to encode simple and moderately complex data struc-

JAVA: INDUSTRY NEWSLETTER

JAVA DEVELOPERS JOURNAL Industry > Newsletter

Monday, December 10, 2001

BROUGHT TO YOU BY...

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

JAVA DUDES

JDJ NEWS DESK EXCLUSIVE

Oracle Targets Top Spot With New Oracle9iAS and JDeveloper

by Rick Ross - Founder of Javalobby.org

(December 10, 2001) – I returned just yesterday from a fantastic trip to San Francisco for Oracle's OpenWorld conference. It wasn't clear what to expect when I accepted Oracle's invitation to give a presentation, but I was delighted to find the show had a high-energy crowd that was very tuned in to Java, especially J2EE and Web services. The whole show was great, but the most interesting and important items came from Oracle, itself - Oracle9i Application Server Release 2 and Oracle9i JDeveloper.

Newsletter

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

WEB SERVICES: INDUSTRY NEWSLETTER

WebServices JOURNAL Industry >Newsletter

Monday, December 3, 2001

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

BROUGHT TO YOU BY...

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

ONLINE EXCLUSIVE: FULL "KEYNOTE PANEL" TRANSCRIPT

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

Barry Morris: I would think we all tend

Newsletter

WSJ News
Next Issue
Subscribe Now!
Privacy Notice

XML: INDUSTRY NEWSLETTER

XML JOURNAL Industry > Newsletter

Tuesday, December 18, 2001

BROUGHT TO YOU BY...

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

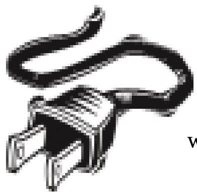
XML NEWS

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

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

Newsletter

XML News
Subscribe Now!
Special Offer!
Privacy Notice



tures in a highly efficient manner. More often than not, XML-RPC will provide you with all the functionality that you need, especially given the natural constraints of wireless devices. For applications that require more functionality, the Simple Object Access Protocol (SOAP) may be in order. In our next article we'll delve into SOAP and provide a detailed analysis of when to choose SOAP over XML-RPC for wireless computing.

Listing 1

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import java.io.*;
import java.util.*;
import org.kxmlrpc.*;

/**
 * @author  Kyle Gabhart
 * @copyright  Kyle Gabhart © 2001
 * @version 1.0
 */
public class MyMidlet extends MIDlet
      implements CommandListener {
    private List              list;
    private Command           exitCommand;
    private String[]          menuItems;
    private Display           display;
    private Alert             response;
    private XmlRpcClient      xmlrpc;
    private Vector            params, xmlArray;

    public MyMidlet() {
        //Initialize the User Interface
        menuItems = new String[] {"Timestamp",
                 "Randomizer", "AddressBook"};
        list = new List( "Select a service",
                 List.IMPLICIT, menuItems, null );
        exitCommand = new Command( "Exit",
            Command.EXIT, 1 );
        response = new Alert("Service Return",
                 null, null, AlertType.INFO);
        response.setTimeout( Alert.FOREVER );

        //Add commands
        list.addCommand( exitCommand );
        list.setCommandListener( this );

        //obtain a reference to the device's UI
        display = Display.getDisplay( this );
    }//end MyMidlet()

    public void startApp() {
        display.setCurrent( list );
    }//end startApp()

    public void pauseApp() {
    }//end pauseApp()

    public void destroyApp( boolean bool ) {
        //clean up
        list = null;
        exitCommand = null;
        display = null;
    }//end destroyApp

    public void commandAction( Command com,
                             Displayable dis ) {
        if ( dis == list &&
                    com == List.SELECT_COMMAND ) {
          switch( list.getSelectedIndex() ) {
           case 0:
            try {
             xmlrpc = new XmlRpcClient(
   "http://www.wsjug.org/servlet/XmlRpcServlet" );
             params = new Vector();
             String serverTime = (String)
xmlrpc.execute( "sysTime.getSystemTime", params );
             response.setString(
                          serverTime.toString() );
             display.setCurrent( response );
            }
            catch ( Exception ex ) {
             response.setString( ex.toString() );
             ex.printStackTrace(); // DEBUG
             display.setCurrent( response );
            }//end try/catch
            break;
           case 1:
      case 2:
            response.setString( "Please download
                             the full sample code);
            display.setCurrent( response );
            break;
          }//end switch( list.getSelectedIndex() )
        }
        else if ( com == exitCommand ) {
            destroyApp( true );
            notifyDestroyed();
        }//end if( dis == list &&
                      com == List.SELECT_COMMAND)
    }//end CommandAction( Command, Displayable )
}//end MyMidlet
```

Download the code at sys-con.com/webservices

2002

JUNE 24–27
JACOB JAVITS CONVENTION CENTER
NEW YORK, NY

OCTOBER 1–3
SAN JOSE CONVENTION CENTER
SAN JOSE, CA

web services EDGE conference & expo™
INTERNATIONAL WEB SERVICES CONFERENCE & EXPO

JDJ EDGE conference & expo™
INTERNATIONAL JAVA DEVELOPER CONFERENCE & EXPO

XML EDGE conference & expo™
INTERNATIONAL XML CONFERENCE & EXPO

JAVA, XML, WEB SERVICES AND .NET TECHNOLOGIES

Fundamentally Improving the Speed, Cost & Flexibility of Business Applications

Who Should Exhibit...

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

These Leading *i*-Technology Shows Feature...

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

Who Should Attend...

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

ONLINE EARLY BIRD REGISTRATION OPENS FEBRUARY 15

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

TO EXHIBIT/SPONSOR
CALL 201-802-3004/201-802-3069
WWW.SYS-CON.COM

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

Web Services Standards

Can there be a consensus?

Many far-reaching claims are being made with regards to Web services. Some in the industry suggest that Web services will make dynamic e-business a reality, will be the next distributed programming paradigm, and will enable the "Holy Grail" of fully distributed Web applications. Component and distributed computing evangelists made those claims before (think CORBA), but these goals remain elusive and unrealized.

As industry observers and participants understand, Web services standards are a moving target. This article will use a layered depiction of the Web Services Stack to examine the more accepted standards at the bottom of the stack, as well as the leading contenders in the murkier waters toward the top. It includes a brief look at some of the competing efforts and several peripheral functional areas that require standardization to enable Web services to achieve its goals. I conclude with some strategies for moving forward in this important area of e-business.

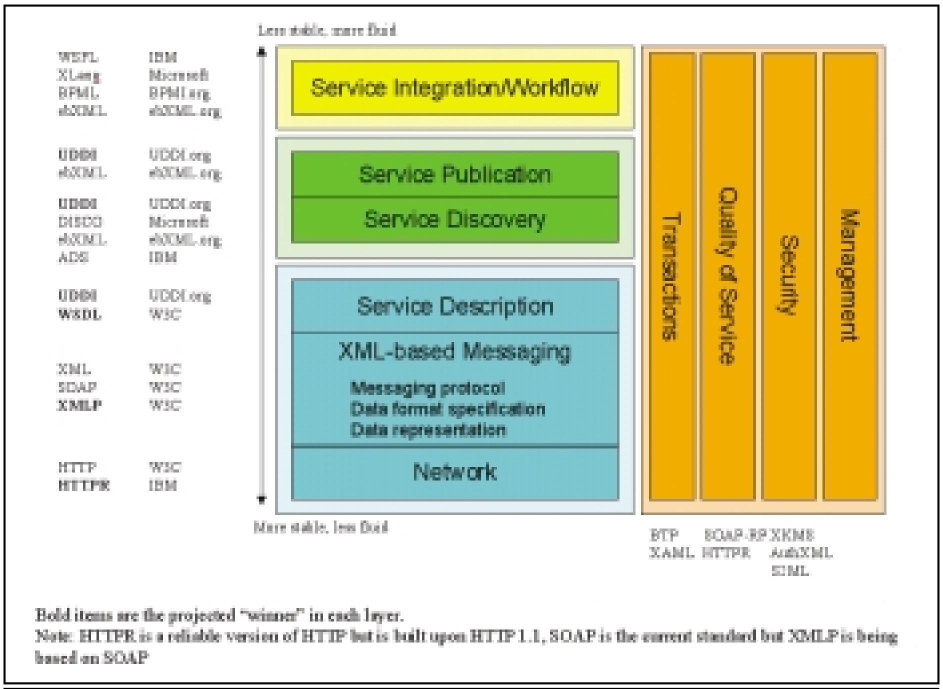
The Players

Almost every major technology company is involved in the Web services standardization effort in one form or another. Many are members of one of the organizations, consortiums, and coalitions that have emerged over the last couple of years. Up to this point, Microsoft and IBM have been driving the more established, lower-level standards. The field becomes much broader and more fragmented when we discuss standards and organizations developing solutions for business process management and workflow.

Currently, the World Wide Web Consortium (W3C) is the primary Web services standardization organization. However, a number of other significant efforts include UDDI.org, OASIS, UN/CEFACT, BPMI.org, and ebXML.

Web Services Stack

The Web Services Stack is an emerging set of open specifications that are either existing Internet standards, or widely accepted specifications that are proceeding through normal procedures to become true standards. It is a layered representation, and the upper layers build on the lower layers. Because the Web Services Stack defines how to construct Web-based solutions, it is fundamental to achieving interoperability (see Figure 1).



Bold items are the projected "winner" in each layer.
Note: HTTPR is a reliable version of HTTP but is built upon HTTP 1.1, SOAP is the current standard but XMLP is being based on SOAP

FIGURE 1 | Web Services Stack

Network

At the bottom of the Web Services Stack is the network layer. Distributed applications require a network protocol, which defines the communication mechanism between two concurrent processes.

Hypertext Transfer Protocol

While the concept of Web services is designed to be protocol-independent, the predominant protocol used today is Hypertext Transfer Protocol (HTTP). The ubiquity of HTTP, coupled with its innate ability to navigate firewalls, makes it the most common Web services network protocol. Use of such protocols as SMTP, FTP, or even TCP is possible, but there are currently only a few implementations that support these protocols.

Recently, IBM published a proposal for a reliable messaging protocol called HTTPR. HTTPR builds its reliability on HTTP 1.1, so it includes the benefits of HTTP while ensuring that messages are delivered to their destinations unimpeded. Reliable messaging is a critical facet of Web services and will undoubtedly

Author Bio

Greg Heidel is a principal architect and the Web services technical lead for Momentum Software. His responsibilities include architecting and designing systems on both the J2EE and .NET platforms and researching the future directions of Web service technologies.

GHEIDEL@MOMENTUMSOFTWARE.COM

appear in some form in the near future, although some may argue whether the network layer is the most appropriate position for its implementation.

XML Messaging

The XML messaging layer includes data representation, data format, and a messaging protocol. The XML messaging layer is generally agreed upon; however, with the ongoing work of the XML Protocol group, it is certain that some refinements are forthcoming.

Data Representation

XML

HTTP is a text-based protocol that is payload-agnostic and therefore lacks a mechanism for representing parameter values in Remote Procedure Calls (RPC). This is where XML, Extensible Markup Language, emerges as an important factor in Web services. XML is a platform-neutral, tagged-data representation language. It allows data to be serialized into a message format that is easily decoded on any platform. XML provides the mechanism for exchanging structured data between network applications.

TABLE 1: WSDL's basic constructs

Connectivity characteristics	
Port	The URL where the Web service is listening.
Binding	The protocol that the Web service is using, generally SOAP.
Logical characteristics:	
Service	The name of the Web service.
Operation	The sequencing of messages that comprise the service.
Implementation characteristics:	
Port Type	A class in an object-oriented language.
Message	A method on a class.
Type	The data types of the message arguments.

XML Namespaces

Extensibility is a key element in the data representation layer. In order to support extensibility, the Web Services Stack needs a mechanism to prevent name collisions and allow a program to process only those elements it cares about. Namespaces offer a simple, universal way to distinguish among identically named elements or attributes. To support extensibility, every element and attribute in XML has a namespace URI associated with it.

Data Format

XML Schemas

Web services require a method to define the data types used in Web services messages.

The XML Schema specification standardizes a vocabulary for describing XML data types. The XML Schema specification also defines a set of built-in primitive data types and a mechanism for establishing the type of an element in an XML document.

Messaging Protocol

Simple Object Access Protocol

The Simple Object Access Protocol (SOAP) is generally agreed upon as the messaging protocol. SOAP is a simple and lightweight XML-based mechanism for exchanging structured data among network applications. It consists of three parts:

- An envelope that defines a framework for describing what is in a message;
- A set of encoding rules for expressing instances of application-defined data types; and,
- A convention for representing remote procedure calls and responses.

XML Protocol

SOAP, in all likelihood, will eventually be replaced by Extensible Markup Language Protocol (XMLP). The W3C created the XML Protocol Working Group to produce a SOAP-like protocol for XML messaging comprising an envelope, object serialization conventions, an HTTP transport binding, and conventions for conducting remote procedure call.

Service Description

One of the goals of Web services is to allow an application to choose between two or more equivalent services in a standardized manner. The rationale is that, at some point, applications will be built from components implemented as network-enabled services and even be able to dynamically select from these services. The service description layer defines the descriptive mechanism needed to provide enough information for a program to make a decision based on criteria such as quality of service, security, or reliability.

Web Services Description Language

The Web Services Description Language (WSDL) describes what functionality a service provides, where the service is located, and how to invoke the service. The WSDL specification defines seven basic constructs that are intended to be language- and platform-neutral and can be somewhat confusing even though they correspond to more widely known computing nomenclature. Table 1 clarifies these constructs:

WSDL is widely supported although it is not yet a recommendation from the W3C. It is currently under review by the XML Protocol Working Group.

Universal Description, Discovery and Integration

The Universal Description, Discovery and Integration (UDDI) specification defines a data structure standard for representing business service descriptions in XML. UDDI provides higher-level business information that is complementary to the information specified in WSDL. There are four basic structures defined by UDDI:

- ***Business entity***: Information about a business, i.e. name, type, etc.
- ***Business service***: A collection of published Web services.
- ***Binding template***: Access information such as a URL.
- ***tModel***: Technical specifications for the service type, such as interface definitions, message formats, message protocols, and security protocols.

TABLE 2: JAX Pack APIs

Java API for XML Parsing (JAXP)	Covers the standard Simple API for XML (SAX), Document, Object Model (DOM), and XSLT.
Java API for XML Binding (JAXB)	A mechanism for compiling XML data type definitions into Java classes capable of reading XML into Java objects and writing them back out again.
Java API for XML-based Messaging (JAXM)	A SOAP-based protocol for sending messages.
Java API for XML Registries (JAXR)	An umbrella specification that provides a unified interface for UDDI and ebXML registries, and conceivably other registries as well.
Java API for XML-based Remote Process Communication (JAX-RPC)	A SOAP-based protocol for requesting operations on remote servers.

Service Publication

Before a potential business partner can make a Web services call, it must first locate a business with the needed service, discover the call interface and semantics, then write or configure its own software to collaborate with the service. The service publication layer defines the mechanism needed to publish a Web service.

UDDI

UDDI has gained a great deal of momentum in the service publication space. The core component of UDDI is the Business Registry, an XML file used to describe a business entity and its Web services. The UDDI Business Registry is a SOAP-based Web service that provides the interface businesses use to publish their Web services to the registry. The registry is operated as a distributed, replicated service. Currently, IBM and Microsoft operate registry nodes.

Service Discovery

Web services are about machine-to-machine communication. To work effectively, this architecture must have an efficient means of integrating Web-based applications and business processes. The Service Discovery layer addresses these needs by defining the means for accessing and consuming service descriptions.

UDDI

UDDI's considerable momentum in this area is due in large part to the backing of Microsoft, IBM, and Arriba. The UDDI Business Registry contains three types of information – white, yellow, and green pages – that businesses can use to discover a Web service.

- ***White pages*** include business name, address, contact, and identifiers, and allow other companies to search the directory by company name.
- ***Yellow pages*** include taxonomies of the types of business and provide for the categorization of companies.
- ***Green pages*** provide a programmatic description about a service and can include items such as references to specifications for Web services. Green pages also allow companies to interface with other companies in the registry using XML and provide a significant key to automating business processes.

DISCO

The DISCO (for Discovery of Web Services) specification, developed by Microsoft, defines an XML-based discovery document format and a protocol for retrieving the discovery document. DISCO enables developers to discover services via an HTTP GET operation. Using the Discovery Document Format (which is also an XML grammar), a discovery document can be sent to a remote server and, if any SOAP enabled Web services exist, receive back a WSDL description of the services provided.

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

BUY THOUSANDS OF PRODUCTS AT GUARANTEED LOWEST PRICES!

FOR ALL YOUR WEB SERVICES SOFTWARE NEEDS

JDJStore.COM

CAPE CLEAR
$445.00 CapeStudio

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

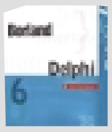
ALTOWEB
$3,540.00 Application Platform Release 2.5

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

CAPE CLEAR
$950.00 CapeConnect Three

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

BORLAND
221980 $2,999.00 Sale Price Delphi



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

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

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

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

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

WWW. .COM OFFER SUBJECT TO CHANGE WITHOUT NOTICE

Special online offer

PICK 4 Subscribe for One Special Low Price

- Wireless Business & Technology
- Java Developer's Journal
- Web Services Journal
- XML-Journal
- WebLogic Developer's Journal
- WebSphereDeveloper's Journal
- ColdFusion Developer's Journal
- PowerBuilder Developer's Journal

SYS-CON MEDIA

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

WWW.SYS-CON.COM

Service Workflow

Business process collaboration and workflow is an essential component for realizing the potential of Web services beyond simple request/response models. This includes the automation and composition of Web services across business boundaries. A critical requirement that will allow inter-enterprise automation and collaboration to succeed is a standardized business protocol for describing these business processes. The service workflow area is very fluid at this time.

Web Services Flow Language

The Web Services Flow Language (WSFL) is a specification for describing business processes. WSFL addresses two types of Web services composition: a business process and a partner interaction. A business process is modeled as a collection of Web services executed in sequence to achieve a particular business goal. A partner interaction describes how the Web services interact with each other. Web services are linked together to indicate the interaction of one Web service with an operation of another Web service's interface.

XLang

XLang is the XML business process language used by Microsoft's BizTalk server. XLang is used to describe the business processes, which are then executed by the BizTalk Orchestration engine at runtime. XLang also allows orchestration of Web services into business processes and composite Web services. One other interesting aspect of XLang is its support for compensating processes – rather than trying to support an expensive two-phase commit protocol, XLang provides a notation for expressing an alternative optimistic model, where actions have explicit compensatory actions that negate the effect of the action.

Microsoft has previously worked with IBM on both the WSDL and UDDI initiatives. And, while they initially created parallel recommendations, it would not be surprising to see IBM and Microsoft jointly agree to submit a proposal to W3C that combines XLang and WSFL in the near future.

Business Process Management Initiative

The Business Process Management Initiative (BPMI) promotes the standardization of common business processes. These processes may span multiple applications, departments, or business partners. The processes may be behind firewalls or accessible via the Internet.

BPMI.org develops open specifications, such as the Business Process Modeling Language (BPML) and the Business Process Query Language (BPQL), that enable standards-based management of e-Business processes with imminent Business Process Management Systems (BPMS).

- The Business Process Modeling Language (BPML) is a metalanguage for the modeling of business processes, just as XML is a metalanguage for the modeling of business data.
- The Business Process Query Language (BPQL) is a management interface to a Process Server, which allows business analysts to query the state and control the execution of process instances managed by the Process Server. This interface is based on SOAP. Process models managed by the Process Repository through the BPQL interface can be exposed as UDDI services for process registration, advertising, and discovery purposes. Both BPML and BPQL are open specifications.

Other Specifications of Interest

A variety of other organizations are contributing to the Web services specifications efforts. A few of the more notable are described below:

ebXML

ebXML, like the Web Services Stack, is an end-to-end stack of protocols and specifications for conducting electronic business over the Internet using standard technologies. ebXML has been discussed as an alternative to Web services and predates the Web services model. However, while there is some overlap between the two models, ebXML focuses more specifically on EDI-style information exchange.

A more likely scenario is the eventual merger of some proportion between the Web services model and ebXML. UN/CEFACT and OASIS, the groups behind ebXML, recently adopted SOAP as the basis of the ebXML-messaging infrastructure. Also, W3C is actively considering the ebXML specification and will incorporate those aspects of the specification that meet their requirements for a standardized Web services architecture.

JAX Pack

JAX Pack is Sun's effort to encapsulate the

RECEIVE $150
DISCOUNT OFF FULL CONFERENCE
WEB SERVICES EDGE REGISTRATION
web services EDGE conference&expo
2002WORLDTOUR
Learn How to Develop
SOAP Web Services NOW!
at a One-Day Tutorial…Coming to a City Near You!

various standards in the Java space. A collection of Java APIs for XML, the JAX Pack is designed to support the standard APIs for Web services including SOAP, XMLP, WSDL, and UDDI.

The various APIs that comprise JAX Pack are outlined in Table 1.

e-Business

In addition to the specifications described above, there are additional critical areas that span all layers of the Web Services Stack. These include security, management, quality of service, and transactions. Businesses will require these additional features before Web services will have the capability to transform their business relationships. Mechanisms for attachments, security, authentication, contract management, quality control, and more will need to follow. Two of the most important issues are security and transactions.

Security

The W3C XML Signature Working Group is chartered to develop an XML-compliant syntax used for representing the signature of Web resources and portions of protocol messages and procedures for computing and verifying these signatures. This working group does not address broader XML security issues such as XML encryption and authorization.

The XML Key Management System (XKMS) is an effort to integrate PKI (public key infrastructure) and digital certificates with XML applications. XKMS relies on the XML Signature specification and on anticipated work at W3C on an XML encryption specification. Other initiatives in this area include S2ML (Security Services Markup Language) and AuthXML, which are being unified under the backing of the OASIS XML Security Services committee.

Business Transaction Protocol

Transactions in Web services have a particularly unique set of requirements. The transaction protocol must be able to address long-running, inter-enterprise business transactions while ensuring the reliable coordination of interdependent workflows.

Business Transaction Protocol (BTP) is an XML-based specification for representing and managing these complex, multi-step transactions over the Internet. BTP provides an open specification for XML message interfaces to support coordination of Web services from different Internet trading partners. In addition, BTP defines a model for defining and managing the status of these interactions to guarantee reliable messaging and completion of a business process, regardless of how long it takes to execute.

This protocol was originally developed by BEA and has since been submitted to the OASIS Business Transactions Technical Committee. This committee has the challenging task of defining requirements, evaluating this submission as well as other technologies, and producing a recommended specification for a business transaction protocol that complements existing Web services standards, specifically the ebXML initiative.

Where Do I Go From Here?

I think that you can now appreciate the vast complexity and fluid nature of the Web services space. The logical question becomes, "Should I even start down the Web services road and, if so, where should I start?"

The short answer is "Yes," but tread carefully. Start by learning and getting familiar with the Web Services Stack, the various specifications, and some of the main vendors and their successful and unsuccessful implementations. Depending on your risk adversity, begin to incorporate these principles, specifications, and standards into either internal or external development efforts. However, always follow common design principles and architect these efforts carefully to accommodate the facts that these standards are still evolving and that change is inevitable.

Conclusion

Will Web services succeed where others have failed? A primary factor in determining the extent to which Web services will succeed is that, for the first time since distributed computing became a mainstream concept, a solution is evolving based on open standards that have the potential to truly support interoperability. Standards identify and eliminate discrepancies, and provide the specificity and transparency that providers and tool vendors need to create interoperable products. How quickly the various players can arrive at consensus on these standards will play a key role in determining the level of success that Web services will enjoy.

Written by Larry Ellingson

Transforming Alliance Airlines' Business Operations

Using SilverStream eXtend

The transformation of Alliance Airlines was a challenging and painstaking experience. The gray hairs on my head and in my beard are firm reminders of the long days and sleepless nights that were required to deliver the end result. I know a few people at SilverStream who feel the same way.

It's important to understand the underlying need to transform a business, whether it's a retailer at a mall or a retailer on the Internet; companies must change and differentiate themselves not only to succeed, but also just to stay alive.

This has been especially true in the airline industry. We've seen deregulation open the world to everyday travel. Airlines began offering deals in hopes of luring the common traveler to fly. Frequent flier programs and advance purchases became the norm as a means to lock passengers into seats. With the emergence of low-cost operators the market became very clouded, as customers saw no difference between airlines: price was the key driver on the majority of seats sold. Mergers, acquisitions, and joint marketing efforts became the key to staying in business; suddenly once-fierce rivals were forced to work with each other just to stay alive. While airlines were fighting to put travelers in seats, the cargo divisions of these airlines were locked in a much more covert fight.

The Changing Business of the Airline Cargo Industry

The emergence of open skies agreements with the UK, Europe, and Far East prompted a globalization of accessible goods to consumers worldwide. We saw a surge in import products to the U.S. as consumers demanded lower prices. Unfortunately, this forced many American companies to move jobs and production onto foreign soil to take advantage of cheaper labor and to be closer to the source of raw material. The buzzword of this period in the mid '80s was "Just-In-Time" (JIT) inventory management. Companies no longer had the capital to house huge inventories and be stuck with excess supplies. FedEx, UPS, and Purolator (systems integrators) emerged as major players in the JIT movement. They laid the infrastructure to provide door-to-door service that enabled goods to be sent directly from manufacturing to the customer's door. A present example of this is Dell Computer, which ships its product directly to consumers.

The cargo divisions of the major international airlines had to do something to stop the bleeding; they were forced to cut costs at any turn. Folklore has it that Lord King, chairman of British Airways, referred to cargo as the ash on the end of his cigar. Outsourcing became the norm. On-airport facilities, high labor costs, ground-handling facilities, and equipment were being shed like snakeskin. The one key thing that many airlines failed to address was information exchange.

Alliance Airlines Fills a Market Need

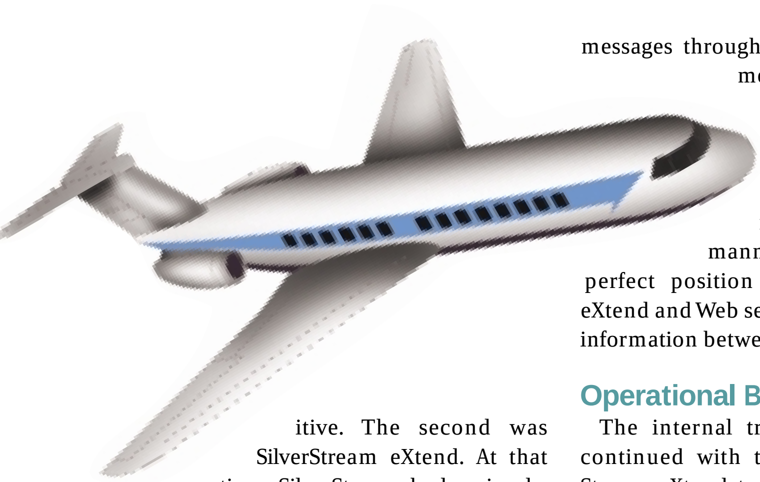
Alliance Airlines saw this as an opportunity to establish itself as a major player in the air freight outsourcing movement. But like many other companies, Alliance needed to manage the tremendous growth it was facing. The company has reported double-digit growth every year since its conception in 1987. Projected statistics for the year ending 2001 have the company handling 20,000 aircraft movements, 700 million pounds, 220,000 consignments, and over 76 million pieces of air freight in more than seven locations throughout the U.S. The underlying philosophy of the company is to build facilities on major international airports and provide both cargo handling and road feeder services as a single source for its clients. Alliance Airlines has an impressive list of clients as well: Korean Air, EVA Airways, Lufthansa, Polar Air Cargo, Air New Zealand, Japan Airlines, and others all outsource their freight operations to that company.

Alliance faced a considerable challenge as these clients began moving massive tonnages in and out of the U.S. There was an immediate need to streamline operations and provide real-time information back to the airlines, as well as other suppliers in the chain. Inland trucking companies, freight forwarders, U.S. Customs, and other agencies all needed to be part of the information flow. Not only were clients demanding quality handling service, they were demanding quality information as well.

The Solution

Alliance soon realized that its DOS-based, Fox Pro system couldn't handle the massive volumes that were being generated so the company began looking for a solution. There was no "off-the-shelf" product that fit the immediate need, and time-to-market was critical. Alliance evaluated two products. The first was an Oracle-based solution. At the time, Oracle didn't have a clearly defined pricing model for Web deployment and the licensing fees were cost-prohib-

Author Bio
Larry Ellingson is a freelance consultant specializing in system architecture and integration, business process reengineering, supply chain logistics, and cultural changes. He previously served as the vice president of information services for Alliance Airlines. Larry has worked with SilverStream since the 1.0 release of the SilverStream eXtend Application Server in 1998.
LELLINGSON@BIZ-PROCESS.COM



itive. The second was SilverStream eXtend. At that time, SilverStream had a simple, clearly defined pricing structure – a per processor license with unlimited users. In our initial meeting, SilverStream fired up its product and quickly put together a small, functional application. At that moment Alliance knew it had found a solution.

SilverStream dispatched three developers to Alliance and using a RAD approach with SilverStream's development environment, we delivered phase one of the project in less than six months. The application quickly became the change agent for Alliance Airlines. The company's internal business problems played into the strengths of the SilverStream eXtend integrated services environment.

Alliance's infrastructure was comprised of many different platforms in the company's back office. The operations system didn't communicate with the billing system, and the billing system wasn't integrated with the GL package. Manual processes; keying and rekeying of data; and slow, inaccurate billing severely limited the vision and goals of the organization. By designing and developing a service-oriented architecture, back office integration became seamless. By the time phase two of the project was rolled out, SilverStream eXtend had enabled full integration of old data into the new environment. We witnessed firsthand the power of XML and Web services with SilverStream eXtend.

While Alliance was undergoing this transformation, the company realized that the other airlines' IT and IS infrastructure was nowhere near as sophisticated as Alliance's. Passenger-facing systems were fairly advanced, but the cargo industry had been left in the cold. Many airlines still think that EDI is new. With this in mind, SilverStream worked closely with Alliance to embed key, event-triggered EDI messages throughout the application. These messages are sent directly to the airline's host system in a predetermined format, allowing a meaningful flow of information in a cost-effective manner. Now Alliance was in a perfect position to leverage SilverStream eXtend and Web services to expand the flow of information between trading partners.

Operational Benefits

The internal transformation of Alliance continued with the deployment of SilverStream eXtend to create a front end for the application. End users throughout the organization had worked in a traditional Windows desktop environment. This front-end allowed them to "float" around and pick what they wanted to work on. Now all relevant services, data, and job functionality are presented to the end user. Work flow practices are invoked for all areas of the business and specific intranet kiosk portal screens are deployed to specific departments. Sales, operations, human resources, and accounting have completely different requirements on a daily basis, and now department heads and business analysts have complete control over the information flow within their divisions. The ease of use makes it possible for IT to take a back seat on day-to-day updates and focus on core development issues.

Interfaces with outside information sources are also no problem. Human resources can transmit employee information captured in an online application to background check companies over the Internet. Flight status of inbound aircraft is now streamed to key operational personnel in the warehouse and on the field via cell phones. Outsourced trucking suppliers communicate with Alliance Airlines via a Vendor Extranet. Status updates, billing, and dispatch are now performed via the Web. The biometric time and attendance application used by Alliance is now integrated with the operating application to produce daily P&Ls by airline and department. Manpower is managed on a proactive basis as opposed to the "bucket brigade" approach used in the past.

I would be remiss if I didn't describe the platform in which SilverStream eXtend is deployed within Alliance. In the early stages, Alliance had fat clients at the desktop, old grinding Pentium 133s (ouch!). All the cities were connected via a Frame relay terminating at their corporate headquarters in Chicago. When SilverStream was first deployed it chewed through those old Pentiums with no trouble; Java apps tend to do that. The decision was made to outsource the complete data and network infrastructure to WORKNET Inc. of Naperville, IL. It turned out to be one of the best decisions we made and the network is now solid as a rock. The data center houses all six of Alliance's servers; they run Citrix over NT and deploy Wyse Terminal thin clients at the desktop. This provided the backbone for the rollout of SilverStream eXtend.

Return on Investment

While it's difficult to put specific numbers on radical business process redesign, there are a few key points that jump out regarding this project. The overall expenditure with SilverStream was approximately $1.5 million. It sounds like quite a bit of money, but if you look at the tremendous growth Alliance is experiencing we probably saved that much alone in manpower.

Manual processes need people to work them, so by automating processes, individuals can now concentrate on core business functions. Capital expenditure spending for a new phone system was postponed, although the phone system Alliance had in place could not keep up with the growth. All areas of the operation had been swamped with service calls but by Web-enabling information, customer service turned into self-service.

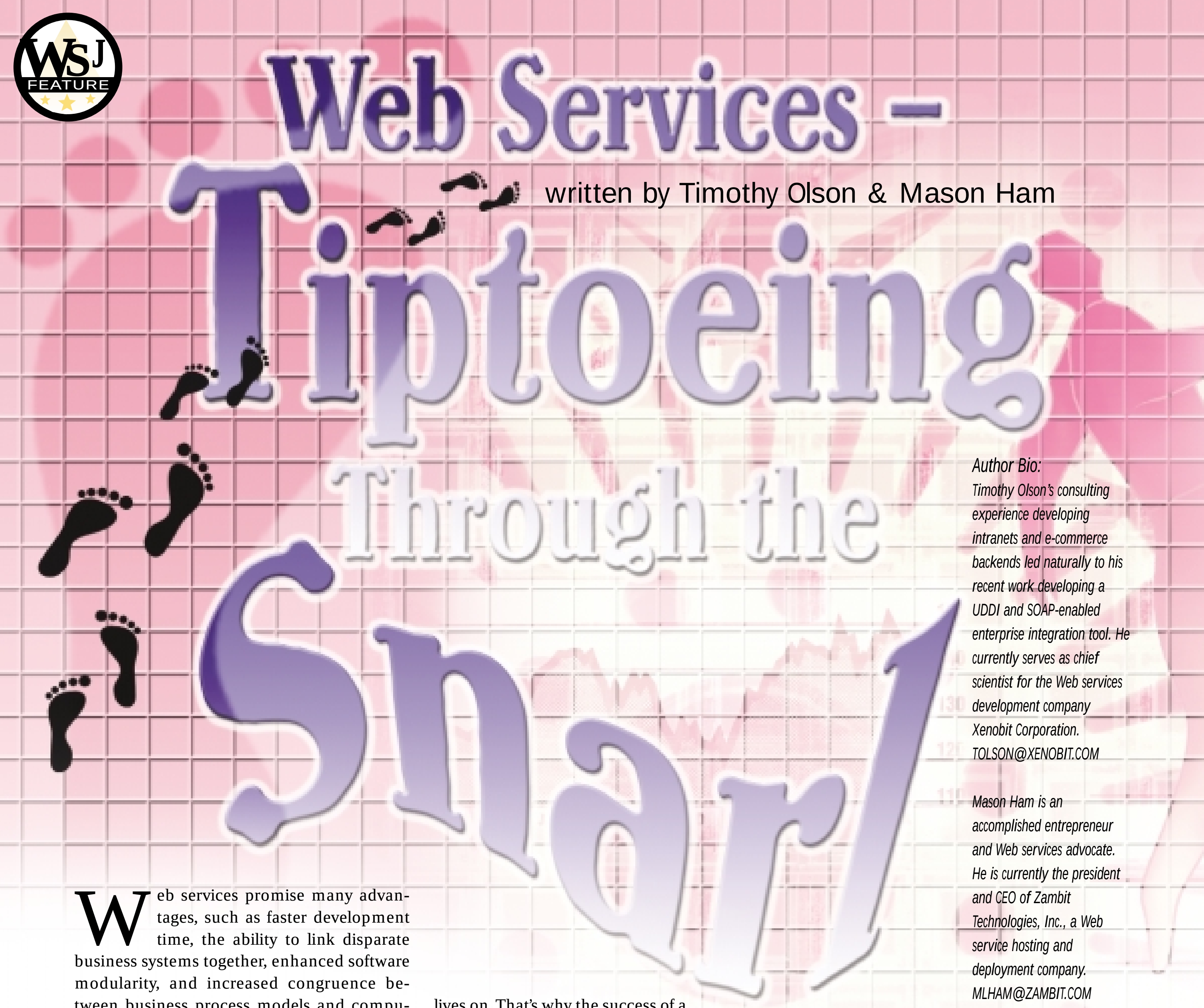
We also uncovered a new revenue stream. By having full control over the import product in the warehouse, Alliance can now strictly enforce terminal service and storage fees. Where freight used to sit for days before collection, the clock starts ticking the minute it's put on location. Customers can check customs clearance and storage fees via the Web prior coming to the airport to pick up goods. The list keeps on going. Needless to say, the investment has been worth it.

Alliance Airlines has leveraged the Internet to transform its business operations and become an industry leader through the close working relationship with SilverStream. The deployment of SilverStream eXtend has ensured Alliance Airlines' place in the logistics supply chain today and into the future.

WSJ FEATURE

Web Services – Tiptoeing Through the Snarl

written by Timothy Olson & Mason Ham



Author Bio:
Timothy Olson's consulting experience developing intranets and e-commerce backends led naturally to his recent work developing a UDDI and SOAP-enabled enterprise integration tool. He currently serves as chief scientist for the Web services development company Xenobit Corporation.
TOLSON@XENOBIT.COM

Mason Ham is an accomplished entrepreneur and Web services advocate. He is currently the president and CEO of Zambit Technologies, Inc., a Web service hosting and deployment company.
MLHAM@ZAMBIT.COM

Web services promise many advantages, such as faster development time, the ability to link disparate business systems together, enhanced software modularity, and increased congruence between business process models and computational architectures. With these in mind, many organizations are considering a Web services–based architecture, but as with any new endeavor, initial implementations meet with practical challenges. Like any new technology paradigm, the goal of utilizing and adapting Web services into a system is to provide value with a low total service cost (TSC). Establishing a concrete TSC is futile because each organization will use its own metrics for determining the total service value. The proper calculation of a TSC, however, will include certain base factors: total number of connections, length of time it takes to integrate, length of continuous uptime, and appropriateness to business needs. To achieve this goal, organizations need to change the way they've traditionally thought about software development. The value of a Web service exists in both the service and the network it lives on. That's why the success of a Web services–based architecture will lie in the design, wiring, and deployment of the system.

This article provides a roadmap to the successful implementation of a Web service–based system. It focuses on the technical issues, including network latency and multiple points of failure, as well as business issues, such as the legality of transmitting data to outsourced Web services and the challenge of dealing with a larger number of software suppliers. With proper planning, the new Web services architecture can provide companies with reduced development time, enhanced software modularity, and a radically reduced total cost of ownership.

The Problem of Network Traffic

A major technical issue in deploying Web services is the performance of the network. Web services generate heavier network loads than traditional client/server architectures. A Web services–based application connects to many services from different servers and even from multiple locations. Any connection between services, which reside on different machines, will increase traffic. A more subtle increase is derived from the nature of the communication protocols used by Web services. In a client/server environment, messages between machines could be custom-designed and binary-based and, therefore, relatively terse. But XML-RPC and SOAP, the protocols commonly used by Web services, are heavier communication protocols because of their reliance on XML, and they increase traffic. Additionally, traffic that leaves the local high-speed network may use a slower Internet connection and may require encryption,

Creating Web services with unprecedented uptimes

which consumes more resources and increases the total payload size. Although none of these costs can be eliminated without sacrificing the abstraction and distribution advantages of Web services, proper planning and design will minimize the impact on the system as a whole.

Smart Service Design

There are several approaches to alleviating high system costs, the most attainable being smart service design. Traditionally, architectures that used a large array of small, refactored components were generally preferred because they allowed systems to grow and change more easily with fewer development dependencies. In the new Web services architecture, not every component needs to be a Web service. The power and value of Web service technology is to connect medium-sized computational service offerings into large-scale computational service offerings, while minimizing the maintenance and scaling costs of running the services in production environments.

Using services that are too small bloats resource consumption, but services that are too large negate the power of composition. When designing a Web service, selecting the optimal size of its problem domain will provide the maximum TSC. Another important design decision is the message content. Using apropos data structures will reduce the amount of extraneous and duplicated information sent across the network. Sending only the information that is absolutely needed not only reduces the size of the messages but also enhances the flexibility and reusability of the service.

When large records need to be sent and the service performs multiple operations on them, it may be worthwhile for the service to cache the data at the request end point. Subsequent requests could then send a reference to the data instead of the entirety. Designing your system with a reasonable number of medium-sized services using well thought-out messages is the most effective way to use the power of the network without being overrun by it.

Connectivity Solutions

The complexity brought on by a Web services architecture manifests itself most directly in the proliferation of network interconnections. Managing the unavoidable increase in the TSC from this architecture requires a two-pronged approach: selecting the correct connectivity solution and the appropriate distribution architecture.

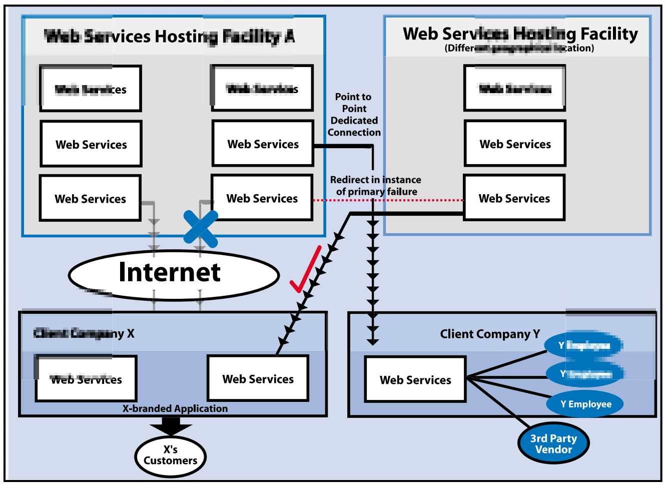


FIGURE 1 | Hosting facility

In selecting the correct connectivity, attention should be paid to the type of network connections used. Given that a high-speed LAN is always more preferable to a distributed WAN, a coordinated hosting facility provides a compelling business case. This facility is a host that specializes in Web services and has hosting agreements with many different Web services vendors (see Figure 1). All the services hosted at the coordinated facility can communicate over a high-speed LAN. Current hardware can move local data quickly, rendering the network portion of the TSC close to zero. By having data travel only on physically secure pipes behind protection systems (e.g., firewalls, proxies, etc.), the need for encryption is reduced, easing the cost and complexity of regulatory compliances.

Systems handling sensitive information not only have to maintain tight physical security but complex business practices as well. Trustworthy Web service hosts and consolidated providers could offer the required compliance in these instances. This is especially important with Web services since data may visit and be exposed at multiple locations before processing is complete.

Clearly, the technique of coalescing Web services into a single location provides great advantages in terms of network usage, but it's not always practical. One facility may not have agreements with all the service vendors a system uses. Multiple facilities may be desirable for redundancy, or maybe a special location handles sensitive information while a cheaper facility is used for the other services in the architecture. Due to the number of possible service fault points created by a geographically dispersed Web services architecture, a close inspection of the TSC must be made.

The use of leased lines can offer many benefits of a single facility while providing the advantages of distributed facilities. Since the data travels on a dedicated connection, the need for encryption is lessened, keeping message sizes small. Leased lines also have high bandwidth. The other major option is to connect across the open Internet, which is easier and cheaper, but also slower, and requires more operational diligence due to possible security, transactional, latency, and transmission concerns.

Services Hosted in Same Location

When deciding how services will interconnect, consider performance requirements of the entire system in terms of maximum acceptable latency, the budget for deployment, the need for security, and the desirability of outsourcing hosting, maintenance, and service management. The best performance can be achieved when all services are hosted in the same physical location. Emerging techniques for Web service optimization and dynamic resource balancing are specialized, proprietary, and generally unavailable to in-house development teams unless they are custom-built.

The Proliferation of Relationships

Just as the use of third-party Web services increases the demand on network resources, there is additional work in establishing and maintaining business relationships with the vendors providing the Web services used. If the utilized Web services are relatively large, standalone applications, there would be fewer services in the system and partnerships would not be much of a problem. When connecting many services into a grand application, however, such as using Web services to model a business process, more vendors are likely to be involved. While UDDI and WSDL make it easier to find, subscribe, and bind to these services, they don't change the need for mutual understanding by the buyer and seller about Service Level Agreements (SLA), costs. If a service is hosted by a vendor and it provides adequate performance for a large application, the agreement may be straightforward, but if the service needs to be hosted in-house or with a third party, special negotiations may need to occur.

Additionally, issues involving the quantification of a TSC for a given Web service can be and complex. Even though the UDDI exposes many services, it doesn't provide a grading system to determine the value of a given Web service. The value of a system can't actually be known because there are no metrics to measure the service offered (in terms of uptime, latency issues, or availability, for example).

Furthermore, the value of a service can't be assumed simply because it was designed by a reputable firm. The TSC of a given Web service is actually affected more by the company deploying the service than it is by the company that created the service. Thus, a given Web service can be deployed by three different companies and have three different TSCs. One of the benefits of Web services is being able to automatically redirect to a backup provider in case the primary fails.

Each Web service used may require additional technical diligence to ensure that the service meets the application's requirements. The architecture may increase a company's resources requirement due to the increase in the complexity and custom development of the integration. If not properly thought out, the hidden costs of managing relationships with many service vendors may hemper an otherwise successful Web services project.

One way to avoid this problem is to use a few fairly complete services. This negates much of the usefulness of a Web service–based architecture but will alleviate the need to have multiple relationships. Broad integration, even interbusiness integration, will require a larger number of interacting services. Modularity and reusability are also sacrificed when using only a few large services. Service failover can also be sacrificed to reduce the number of interactions necessary when the primary provider can give adequate uptime assurances. Initially, using only a few services is an easy way to approach the technology, and existing systems can be slowly migrated.

As Web services become more widespread, the problem of relationship proliferation will have to be addressed. An important impact of this new architecture is the need for closer and more consistent communication between a company's technical staff and the in-house partnership staff. Web services that are deployed outside of the organization are in fact partnerships. Staff, dedicated to vendor partnerships, can take on additional responsibilities related to Web service vendors, but they must work closely with the development team to ensure that every provided service meets their technical requirements.

Additionally, the technical team needs to assume the responsibilities for scoping a given Web service and its integration. Good communication between the Web service provider and a partnership staff member will add to the success of the Web services–based system.

Legal Issues and Data Regulations

Using Web services increases the load on the computer network, the number of relationships that need to be managed, and also the corresponding legal work that must accompany the myriad agreements. Beyond this lurks another challenge in deploying a Web services–based system. Because of the way Web services distribute data to each other – to multiple locations, and even to multiple businesses – careful consideration must be given to the way data is handled, stored, and transmitted to ensure compliance with privacy laws and data exchange regulations.

U.S. and international privacy laws are still evolving, and an ill-conceived Web services system may become unlawful, even if currently compliant. European laws are much tougher and may exclude some of the most obvious ways to connect Web services together. Industry-specific regulations must also be considered. A healthcare system might not be allowed to send patient data to a Web service hosted outside the company. A financial system may have special authentication requirements.

Additionally, current privacy statements may exclude the transmission of personally identifiable information. What if a remote Web service caches this information? Has it been given out to another company? By being distributed among multiple service providers, a Web services–based application encounters more than the usual amount of legal issues.

Tiptoeing Through the Snarl

Complying with regulations requires a high awareness and an active posture. The primary legal complication with Web services is in the way data moves

WebServices JOURNAL .NET J2EE XML

COMING IN THE
FEBRUARY ISSUE

FOCUS ON MICROSOFT .NET

- **Microsoft .NET: Past the Hype to the Reality** by Sean Rhody
- **MS Gets It At Last!** by Paul Lipton
- **Mobile .NET Web Servces, part 2** by Derek Ferguson
- **J2EE vs. .NET: Friends or Foes?** by Scott Dietzen
- **Getting Greater Business Value from a Web Service** by Paul Hernacki and Keith Landers
- **Bringing Buyers and Sellers Together** by Eric Lynn

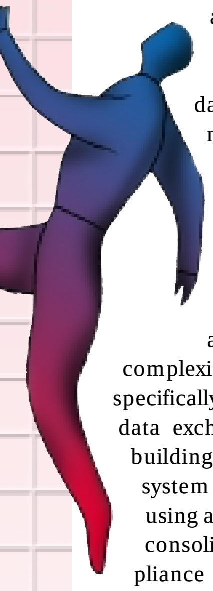
WSJ ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
Apress	www.apress.com	212-533-3503	23
BaseBeans Engineering	www.basebeans.com	866-460-2320	65
BEA eWorld	www.bea.com/events/world/2002/	404-240-5506	35
Borland	www.borland.com/new/k2/2014.html		19
DataMirror	www.datamirror.com/resourcecenter	800-362-5955	25
Infragistics	www.infragistics.com	800-231-8588	29
IONA	www.iona.com		4
Java Developer's Journal	www.sys-con.com	800-513-7111	37
JavaOne	http://java.sun.com/javaone/	888-886-8769	47
JDJ Store	www.jdjstore.com	888-303-JAVA	39, 53
shinka technologies	www.shinkatech.com/register7/	800-SHINKA1	27
SilverStream	www.silverstream.com		67
Software AG	www.softwareagusa.com/XMLWeb	877-SAG-4XML ext. 133	13
Sonic Software	www.sonicsoftware.com		11
SpiritSoft	www.spiritsoft.net/downloads		9
Sun Microsystems	www.sun.com/sunoneinfo		2, 3
SYS-CON Custom Media	www.sys-con.com	925-244-9109	51
SYS-CON Media	www.sys-con.com	800-513-7111	53
Web Logic Developer's Journal	www.sys-con.com	800-513-7111	61
Web Services Edge Conference & Expo	www.sys-con.com	201-802-3069	49, 54, 55
Web Services Journal	www.sys-con.com	201-802-3024	31, 43
WebSphere Developer's Journal	www.sys-con.com	800-513-7111	65
Wireless Edge Conference & Expo	www.sys-con.com	201-802-3004	63
XML Global	www.xmlglobal.com/newangle	800-201-1848	68
XML-Journal	www.sys-con.com	800-513-7111	60

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.

through the system to multiple service vendors. If all services can be hosted in-house, then the issues are no different than in a traditional computing system. If outside services are used, however, all current regulations must be reviewed with the Web services architecture in mind. What data will be sent where? Where and when will it be stored? Who has access to the data during any part of the process?

Web services vendors should be required to produce an explicit statement about how their service stores or manipulates the data, including second-level Web services they use. With these considerations in mind, review existing privacy statements, data use agreements, relevant laws, and any other possible restriction on the use, transmission, or storage of the data the Web services system will manipulate. Even after the system is confirmed compliant and deployed, the liquid nature of current regulations requires diligent attention to emerging changes.



The concept of Web services can actually help manage this legal complexity. By using a Web service, which specifically understands and complies with data exchange regulations, the burden of building and continually changing your system for compliance is reduced. When using a Web service host or dealing with a consolidated provider, additional compliance claims might be made about the interactions between multiple services in the offered group. Using such a suite of services further reduces the legal analysis needed.

Multiple Points of Failure

A final challenge in planning a Web services–based system is coping with multiple points of failure. If a system or application is developed by subscribing to and wiring together multiple Web services, what happens when one of the services goes down? It could affect the entire process. What about maintaining transactional integrity? If the first stages of the process are completed successfully but later stages fail, rollbacks may be necessary on parts of the system that never displayed an error, and any data that didn't complete the entire process needs to be retried – or else it will be lost.

Fortunately, these difficulties are not unique to systems composed of Web services, and reliability techniques are well known. Redundancy is the keyword. Multiple independent network connections, automatic rerouting to duplicated hardware components, and even redundant power sources will multiply uptimes. Of course, building such reliable infrastructure correctly is tricky and definitely carries a cost, but there are many hosting companies with the required specialties. No matter how reliable the hardware system is, there will be failures. What then?

Web Services to the Rescue

Beyond the usual precautions, a Web services architecture can provide additional contingency operations. Because of the modular nature of a well-designed Web services system, when one service does fail, the request can be rerouted to an identical service on a different machine, at a different location, or even to a different vendor entirely. If done correctly, multiple points of failure can become multiple points of reliability. To achieve this reliability, an early understanding of what services need to be duplicated has to be addressed early in the service development cycle. Existing standards, such as WSDL, have mechanisms for supporting failover, but implementation requires either some kind of orchestration software or special code written into each module that calls a Web service. If the effort is given, however, even more clever resource management techniques are opened.

Service Spawning

If a system is built to dynamically redirect Web service requests, then a technique called *service spawning* can be used to provide redundancy, relieve bottlenecks, and use hardware resources efficiently. Service spawning is the technique of starting and stopping services on the fly, as they are needed, where they are needed. If one machine goes down, the Web service it was hosting can be automatically started on another machine, and the requests can be redirected. Similarly, new service instances can be started if the response times drop below a certain threshold, automatically clearing up bottlenecks. Using service spawning, a given Web service will be available as long as some hardware, a network connection, and the spawning software are functional.

Another way a Web services–based system can turn multiple locations to its favor is by using a service request proxy. Instead of invoking a Web service directly, the request is queued on a proxy. If the Web service goes down, the proxy retains all the request information and can redirect to another service. This also helps maintain transactional integrity, although response times may be impacted if the proxy is not colocated with the Web services themselves. If it's colocated, then it potentially has the same vulnerability as the resources running the services. If a service goes down, does the proxy go down as well? If the proxy goes down, is there another route to the service? One method is to build the forwarding queue into the software that makes the Web services calls. This eliminates the impact on response time and avoids adding maintenance on a separate piece of software. If orchestration software is used to build the Web services system, it should have queuing and redirection built in.

Unprecedented Uptimes

Achieving success with Web services is not straightforward and requires attention to many aspects of the project. The complexity of the Web services architecture is not in any one aspect of the system, but in the broad range of technologies and understanding that's required to deliver this architecture. The complexity of obtaining a good TSC can be a daunting task. But by taking a systematic approach to the problem and decomposing it into the four major aspects – transmission latencies, service refactoring, architecture and bindings, and deployment – it's possible to provide some benchmarks to say that one process is better than another.

Network latencies from remotely distributed systems can be handled by collecting services into a single facility. Taming the large number of software partners might initially involve reducing the number of services used, but long-term solutions are to be found from companies that coordinate a variety of Web services into a single billing and contact point. A smart architecture uses just the right amount of Web services and can help reduce network load and mitigate the number of software partners. Legal concerns revolve around the transfer of data between many Web service providers, and privacy statements, privacy laws, and industry regulations should all be reviewed with a Web services system in mind. Services that handle sensitive information may be hosted in-house or specially designed to handle data in a compliant manner.

Finally, a Web services system without proper redundancy will be less reliable than traditional systems, but if the correct techniques are used, systems using Web services can achieve unprecedented uptimes. The highly touted advantages of Web services are possible but not automatic. Careful consideration of the preceding factors can help guide a project through to a successful completion with a substantially lower TSC.

wireless EDGE conference&expo

Plan to Exhibit

Provide the Resources To Implement Wireless Strategy

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

INTERNATIONAL WIRELESS BUSINESS & TECHNOLOGY CONFERENCE & EXPO

CONFERENCE & EXPO

CONFERENCE & EXPO

Shaping Wireless Strategy for the Enterprise

Santa Clara, CA

May 7-9, 2002

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

Plan to Attend the 3-DAY Conference

WHO SHOULD ATTEND

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

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

WWW.WIRELESSEDGE2002.COM

SHAPE YOUR WIRELESS STRATEGY... SAVE THE DATES!

Conference Tracks

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

FOR EXHIBIT & SPONSORSHIP INFORMATION PLEASE CALL

201 802-3004

SPEAKER PROPOSALS INVITED

EXCLUSIVE SPONSORSHIPS AVAILABLE

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

SYS-CON MEDIA

SYS-CON EVENTS

WWW.SYS-CON.COM

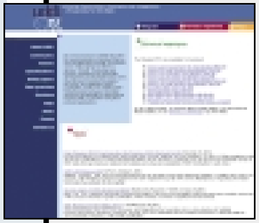
WSJ News

Systinet Releases Enterprise-Class Web Services Platform

(Cambridge, MA) – Systinet Corporation, a provider of Web services infrastructure software, has released WASP Server Advanced 3.0 for Java, a complete platform for the development and deployment of Web services. WASP Server Advanced includes essential enterprise-class features such as remote references, full J2EE integration, and security based on GSS-API/SPKM. It supports all the latest versions of the Web Services standards. WASP Server Advanced 3.0 for Java is available immediately and is free for development and test purposes. It can be downloaded from Systinet's Web site, www.systinet.com

UDDI Project Launches Next Version of UDDI Business Registry

(Orcas Island, WA) – The Universal Description, Discovery, and Integration (UDDI) project has announced public availability of the UDDI Business Registry v2 beta, continuing its mission to deliver advanced specifications and real-world implementations to the developer community. Hewlett Packard Company, IBM, Microsoft, and SAP have also launched beta implementations of their UDDI sites that conform to this latest specification.



The UDDI v2 specification expands UDDI functionality to enhance support for deploying public and private Web service registries. In addition to taking advantage of the public UDDI Business Registry sites, enterprises can also deploy private registries to manage internal Web services. Access to internal Web service information may also be extended to a private network of business partners. The new features include improved programmatic access to UDDI Data, stronger business relationship modeling, and enhanced availability of UDDI information to the end user.
www.uddi.org

Resonate Commanders Solutions 2.0 Now Available

(SUNNYVALE, CA) – Resonate Inc., has announced the immediate availability of Resonate Commander Solutions 2.0, active service-level management solutions for e-business and Web services, the only solutions that allow enterprises to truly manage and maintain service levels for e-business applications and Web Services. With the ability to track and manage services automatically, in real-time, businesses can realize high application availability and establish predictable performance objectives to meet specific business initiatives. Resonate Commander Solutions 2.0 proactively manage the user experience while resolving problems in the e-business infrastructure.
www.resonate.com

AltoWeb Ships Application Platform 2.5

(Palo Alto, CA) – AltoWeb, Inc., has announced the availability of the AltoWeb Application Platform Release 2.5, the first packaged alternative to lengthy, custom, and complex coding of J2EE, XML, and Web Services. This release delivers up to a 10x gain in productivity and time to market with J2EE-compliant application servers and Web services. The AltoWeb Application Platform provides a proven application architecture with packaged logic, and a streamlined approach to application assembly, testing, deployment, and management. The company has posted a free downloadable 30-day evaluation copy for BEA WebLogic Server, IBM WebSphere Application Server, and JBoss/Tomcat on its Web site.
www.altoweb.com.

The Face of J2EE and Web Services

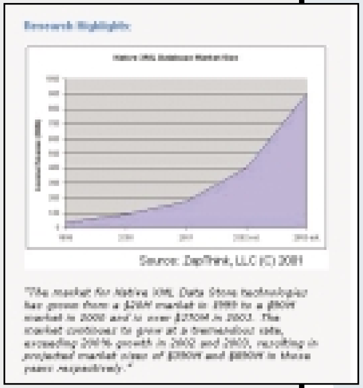
Bind Systems Releases BindPartner Platform

Orlando, FL, December 11th, 2001 -- Bind Systems Ltd., a business process-driven Web services platform vendor has announced the availability of the BindPartner Platform. This platform delivers a business-centric environment for developing, executing, and managing collaborative business processes within and between organizations.

The J2EET-based BindPartner Platform supports leading Web services standards and provides business integration through JavaT, EJB, JAXM, and JMS adapters.
www.bindsys.com

ZapThink Report Shows Market for Web Services Technologies to Expand to $15.5 Billion in 2005

(Waltham, MA) – The market for Web services platforms, application development suites, and management tools will expand from a $380 million (US) market in 2001 to over $15.5 billion (US) in 2005 according to a report released by ZapThink, LLC, an XML-focused industry analyst group. The report highlights recent Web services developments and identifies the different components of the Web services-related industries from a market-wide perspective.



"The market for Native XML Data Store technologies has grown from a $26M market in 1999 to a $80M market in 2000 and is over $170M in 2001. The market continues to grow at a tremendous rate, exceeding 200% growth in 2002 and 2003, resulting in projected market sizes of $390M and $850M in those years respectively."

The ZapThink Q4 2001 Web Services Technologies and Trends Report provides a comprehensive view of the current Web services landscape, including market overview, Web services architectures and definitions, market segmentation, and a detailed analysis of constituent market segments, market challenges, and vendor profiles. A summary of the report may be found on their Web site at www.zapthink.com.

SilverStream Software Releases SilverStream eXtend JEDDI Registry

(Billerica, MA) – SilverStream Software, Inc., has announced that SilverStream eXtend JEDDI (Java Enterprise Discovery, Description and Integration), a comprehensive UDDI implementation, is available for free download at http://software.silverstream.com. This free developer edition of the product encourages developers to discover the benefits of UDDI for publishing, managing, and sharing Web Services internally across development projects.

SilverStream eXtend JEDDI, currently available in beta, is an open implementation of the UDDI v1.0 specification and can be used for public Web services development and testing as well as private UDDI deployments. It includes administration tools, core data, and optional test data to help developers get started quickly with UDDI.
www.silverstream.com

BaseBeans Engineering

www.basebeans.com

The Triumph of Hype Over Experience

WRITTEN BY

Anne Thomas Manes

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

ATM@SYSTINET.COM

Hype is a very useful marketing tool. You come up with a new idea, something with real potential. You go out and raise awareness, you evangelize about how this new technology will revolutionize business. If you market it well, you create a buzz. The next thing you know, you've got lots of people talking about it. New businesses start popping up. Money starts to flow. Suddenly you're on your way to endless riches...at least for a little while.

But there's a problem with hype. If you're not careful, the idea will get exaggerated. Expectations can get totally out of hand. What starts out as a good idea can turn into something totally unattainable. Then, if you don't deliver on the hype within an unrealistically short time frame, the public begins to doubt that the technology will ever work. And then the bubble bursts.

Crashing and Burning

There are lots of examples of exaggerated hype leading to "crash and burn." Look at peer-to-peer (P2P) technology, for instance: P2P was all the rage 18 months ago. Now it's hard to find anyone concentrating on P2P. The money has dried up. It's not that P2P is a bad idea – in fact, it's a great idea. But it's hard to create a viable business model based on P2P technology alone. I'm still waiting for someone to use P2P to solve a real business problem that can't be solved more easily using some other technology.

You might also recall that 18 months ago, Bluetooth was also the talk of the town. Every device imaginable would support Bluetooth-based wireless communications. To date, there are only about 350 products that are Bluetooth-qualified, and very few of them are shipping. The technology just hasn't lived up to its promise of offering a simple, robust, low power, short-range wireless solution at an affordable price. The public has just about given up hope that it will ever happen.

Whatever Happened to 3G?

And where are WAP, third generation wireless technology (3G), and the ubiquitous "Wireless Web"?

Wireless LANs are great, SMS messaging is really useful. But whatever happened to always-on, totally mobile, broadband, Internet access? I remember swallowing the hype that said that 3G, operating at speeds of up to 2MB/sec, would be fully deployed by 2001. People would be able to use a wireless handset to interface with personal information management systems and corporate applications from anywhere in the world.

Maybe I just live on the wrong continent, since I'm told that 3G is indeed being deployed in Asia and in Europe. But here in the States, we're just barely getting access to SMS and GPRS. I'm also definitely having trouble grappling with the cost of always-on wireless Web service – particularly when you consider how limited that service is. It's rare to find someone using a cell phone, or even a two-way pager, to interface with corporate applications.

Will Web services go the way of P2P, Bluetooth, and the wireless Web?

I'm worried that Web services technology might also be in danger from over-hype. I cringe every time I hear someone telling me that I'll be able to use the public UDDI registry to programmatically discover a new materials supplier and dynamically place an order without human-to-human interaction. It's just not going to happen. It's not a problem with the technology. The technology can do it today. It's a matter of business behavior. You just don't order materials without verifying the validity or quality of the supplier.

Business Model Sorely Needed

Okay. So what about consumer-oriented services? How about traffic reports or restaurant guides or sports scores? As a consumer, I'm probably willing to bypass the due diligence process for this type of service provider, so perhaps I'd be willing to dynamically discover and invoke one of these services. But then I ask myself, who's going to pay for these services? Will they be sponsored by advertising? That business model doesn't seem to work very well anymore. Will the consumer have to pay for them? If so, dynamic discovery and invocation doesn't seem quite so viable anymore.

The Future Will Come...Just Not Yet

So perhaps it's time to rein in our expectations a little bit. Web services technology is very powerful. It allows us to integrate our systems more easily than ever before, and thereby helps us improve efficiency and reduce costs. It can offer real business benefits. Let's take advantage of those benefits now, rather than getting distracted by some unrealistic future fantasy.

Perhaps someday – maybe even in less than five years – I'll be able to walk into a room and use my Bluetooth-enabled mobile viewer to dynamically discover a local video-on-demand service that uses a peer-to-peer content delivery network to stream the video to my viewer over 4G wireless broadband, while automatically charging the usage fee to my financial provider as defined by my identity service.

It's always fun to dream about the future. But let's just make sure that we keep things in perspective.

SilverStream

www.silverstream.com