

Java Web services: The year ahead in Java Web services

Learn about the changing landscape of Web services for Java development

Skill Level: Introductory

[Dennis Sosnoski \(dms@sosnoski.com\)](mailto:dms@sosnoski.com)

Consultant

Sosnoski Software Solutions, Inc.

09 Feb 2006

The coming year is bringing dramatic changes to the Web services landscape. For Java™ developers, these changes will include both new Web services frameworks and new layers of functionality built on top of Web services. In this first part of his "Java Web Services" series, Dennis Sosnoski looks at the coming changes and plots a course for readers.

The coming year, 2006, is going to be a banner year for Web services in general, and for Java Web services in particular. New third-generation frameworks are being unveiled, which offer much better support for doc/lit SOAP as well as potential performance improvements. At the same time, the froth of WS-* standards is finally starting to settle into a common set of interoperable layers that extend SOAP and WSDL to support core enterprise requirements.

In this first part of my series on Java Web services, I review both the state of Web services today and the major changes coming in 2006, with an overview of how the new frameworks and technologies relate and interact. The following articles will look at many of these frameworks and technologies in-depth, with the goal of keeping you informed of the latest developments in the field and aware of how they can be used to aid your programming projects.

Setting the stage

It's been over six years since the publication of the SOAP 1.0 specification. Developers had been exchanging XML messages over Internet protocols long before the SOAP specification was published, but the addition of SOAP promised to formalize this technique and allow better interoperability. SOAP also provided the hooks for easy extensibility, so that higher-level infrastructure functions could be added to enhance XML message exchange in the future. The WSDL specification, published soon after SOAP, added a standardized representation for Web services metadata. Major software vendors quickly saw the potential of the SOAP and WSDL combination, and for the next few years it looked like SOAP Web services were the wave of the future.

SOAP and WSDL challenges

Despite the rapid uptake of SOAP+WSDL across the industry, there have been some problem areas that have prevented SOAP from achieving the full success many expected. The first such problem area is in interoperability. Even though the promise of interoperability was one of the cornerstones of SOAP's appeal, the delivery has been lacking. Initially this was due to a focus on the rpc/encoded style of Web services (also known as rpc/enc), where an object model is serialized to XML and then reconstructed on the receiving end. This automatic serialization/deserialization makes rpc/enc easy to use (as long as you stay with the relatively simple data structures it supports), but results in XML that is unusable for any other purpose. Worse still, differences in language and platform support resulted in many incompatibilities between implementations.

The accepted best practice for Web services is now to forego the use of rpc/enc style in favor of the document/literal style (doc/lit). In doc/lit, XML message formats are defined by a W3C XML Schema definition. In theory, this should eliminate any problems with interoperability, since the schema instance defines the actual structure of the XML, and it's up to each platform to process that XML as appropriate. In practice, different levels of support for the extremely complex W3C Schema standard has resulted in yet another set of interoperability problems.

Both the earlier rpc/enc and the more recent doc/lit interoperability issues have been made even worse by a lack of acknowledgement. This is especially acute for doc/lit, where frameworks support different subsets of the schema standard without listing the missing features. Even where different frameworks claim to support particular schema features, the implementations are often incomplete and result in interoperability problems when using those features. Part of the reason for the move to doc/lit was the desire to make use of enterprise or industry standard schemas. Such standard schemas are generally designed without concern for their use in Web services, so they often use features which are poorly supported by SOAP frameworks.

Another problem area for SOAP Web services is the continuing confusion over infrastructure extensions to basic SOAP processing -- added layers of processing

which can be applied across a range of Web services. The design of SOAP allows for the easy addition of such extensions, but these extensions are generally only useful if they're standards supported by multiple frameworks. That requires industry-wide cooperation, which has been difficult to achieve. Even the most fundamental extensions, for features such as attachments and security, have taken several years to develop and are still not supported by all frameworks.

Resistance to SOAP

While the problems with interoperability and standardization detailed in the last section have limited the usefulness of SOAP Web services, the SOAP frameworks themselves are often complex to configure and difficult to use. This combination of limited benefits for substantial complexity has caused many developers to look for simpler alternatives to SOAP. Most of the SOAP resistance movement has coalesced around a technology known as REST. Strictly speaking, REST is a formalization of the basic rules of the HTTP protocol which can be applied to Web services. In practice, the REST movement often leaves aside formalization and encompasses anything that transfers XML documents over HTTP without a SOAP wrapper, basically having co-opted the idea of direct XML document exchange that predated SOAP.

REST is much less ambitious than SOAP. REST is inherently limited to the use of HTTP as a transport layer (though similar approaches could be used for other transports), while SOAP is in theory transport-agnostic (though it has only been widely deployed using HTTP transport, so far). REST does not include any direct way of adding infrastructure extensions -- but until SOAP really starts to deliver on such extensions, this limitation can be seen as a moot point.

Because REST is less ambitious than SOAP there's generally no need to use any framework code to implement a client or server, so developers don't need to deal with the complexities of a framework. The tradeoff is that they *do* need to implement the HTTP and XML handling directly, but many developers have already become comfortable with using these technologies. The direct handling of XML can even be an advantage, since it allows developers to choose from a wider range of processing choices than what is offered by the SOAP frameworks.

So is it time to just bury SOAP and move to the simpler REST alternative? This is probably a practical alternative for many forms of Web service applications, so I won't dismiss the idea out of hand. However, there are other applications, particularly at the enterprise level, that require the sort of added infrastructure and transport agnosticism that SOAP still promises to deliver. Moving to REST would mean that these applications would need to directly implement functions such as security, transactions, and coordination, rather than having these functions provided through a framework. Most enterprise applications would probably chose to avoid Web services altogether rather than go through this amount of effort.

But just like in a movie, even as the situation looks really grim for SOAP a new hope is arising. This hope springs from a new generation of frameworks now arriving on the scene. These frameworks are set to finally deliver on the potential of SOAP, making possible a whole new range of SOAP Web services applications while also dramatically improving the interoperability of doc/lit Web services.

The importance of Indigo

Even though this is a Java technology series, the very first new framework I'm going to mention comes from the archrival to Java technology for the hearts and minds of developers: Microsoft® .NET. This new framework is Windows Communication Foundation (WCF), also known as Indigo. Indigo was originally intended to be part of the "Longhorn" release of Windows scheduled to ship for the last several years, but Microsoft has announced that in the form of WCF it will also be made available for older versions of Windows. WCF is expected to replace the older .NET framework once it becomes available.

What makes WCF important to the world of Web services is the sheer weight of numbers that Microsoft controls through its effective ownership of the desktop (not *complete* ownership -- like a growing number of people I use Linux® for everything, and Macs are also popular -- but 90%+). This ownership of the desktop means that when Microsoft comes out with a new framework, it has a major impact. The technologies supported by Microsoft automatically become targets for most other frameworks to support, and those left out by Microsoft may become second-class citizens usable only when Microsoft systems can be excluded from both the clients and the servers.

With WCF, Microsoft is adding major new technologies to the basic .NET platform (though several are currently available in the form of the WSE 3.0 add-on to the basic .NET). These technologies include XOP/MTOM, WS-Addressing, WS-Trust, WS-SecureConversation, WS-ReliableMessaging, WS-Coordination, WS-AtomicTransaction, and WS-Policy. XOP and MTOM are new standards supporting the transmission of binary data included in a SOAP message as an attachment, which should finally lead to interoperable attachments across all major SOAP frameworks (previously Microsoft only supported an attachment technology called DIME, while the majority of frameworks supported an earlier Microsoft proposal called SwA). WS-Addressing provides a standard format for message identifiers, target addresses, and actions; the identifiers part is important as a requirement of several of the other technologies, while the addresses and actions parts are needed to support alternative transports (other than HTTP) and asynchronous operations. WS-Trust and WS-SecureConversation complement the older (and already widely supported) WS-Security with support for more-performant symmetric encryption. WS-ReliableMessaging supports message delivery and sequencing guarantees. WS-Coordination manages sequences of operations in a distributed network of Web services. WS-AtomicTransaction supports transactions over SOAP with a distributed two-phase commit protocol. Finally, WS-Policy defines

extensions to WSDL to let services state their requirements for using all of these technologies. These WCF technologies represent most of the support services necessary for building enterprise applications with Web services.

If the technologies included in WCF *do* become widely supported and interoperable we'll have an excellent reason for building enterprise Web services around SOAP. Right now the odds look good that we'll see this happen. Most of the major SOAP frameworks were represented at a recent WCF Interoperability Plug-fest sponsored by Microsoft in November, 2005, including the Java alternatives I'm going to focus on in the remainder of this article. These early testing results were very limited, and there are some real problems along the way to full interoperability (including the particular version of still-changing WS-* standards to be supported), but the direction in the industry is clearly to support the WCF technology choices as a core for the next generation of SOAP frameworks.

Sun and the Java standards

JAX-RPC 1.0 was the original standard for Web services in Java. Although JAX-RPC was designed with the idea that different protocol implementations might be used for the actual Web service implementation, in practice it has only been used for SOAP services. Several different implementations of JAX-RPC have been developed, with the most widely used implementation probably being the Apache Axis framework, followed by the Reference Implementation included as part of the Java Web Services Developer Pack distributed by Sun Microsystems.

At the time JAX-RPC 1.0 was developed, many people in the industry thought the `rpc/enc` style of SOAP would be the most convenient and useful form of Web services. The JAX-RPC 1.0 specification required basic support for both `rpc/enc` and `doc/lit` styles, but did not require support for many features of schema. This had the unfortunate side effect of making `doc/lit` SOAP (which is based on schemas) effectively a second-class option.

JAX-RPC 1.0 also took a somewhat limited view of the functionality of Web services. As suggested by the name, the intent was to support RPC (Remote Procedure Call) operations using XML. Java already had an existing technology for RPC calls between Java applications, in the form of RMI (Remote Method Invocation). The specification team chose to model JAX-RPC on the existing RMI interface. This model makes for a reasonably close match as long as you're working with `rpc/enc` SOAP using request-response operations over HTTP, but doesn't map well to asynchronous operations or to other transports. By the end of 2003 it was clear that a major revision of JAX-RPC was needed to handle these and other issues, so Sun formed an expert group to develop a JAX-RPC 2.0 specification.

JAX-*

The main goals of the JAX-RPC 2.0 effort were to update the standards support mandated for JAX-RPC 1.X, build on Java 5 features such as annotations and generics, improve messaging support (including asynchronous operations and transports besides HTTP), and improve schema support by using JAXB 2.0 data binding in place of JAX-RPC 1.X's simple (but very limited) built-in binding. Partially to emphasize the extent of the changes, the name of the successor standard was changed to JAX-WS 2.0. JAX-WS 2.0 is now available in near-production release form, with the production release expected before mid-2006.

JAX-WS 2.0 succeeds in meeting the goals set for the follow-on to JAX-RPC 1.X, and even adds in some goodies like limited REST support. Some developers may have problems using JAX-WS 2.0 due to its heavy use of annotations and other Java 5 features (which makes it unusable with older JVMs), but for many developers the reliance on Java 5 features will be an advantage. A larger concern may be that JAX-WS 2.0 does not support any alternative to annotations for Web services configuration, which may limit the flexibility and long-term advantages of the framework.

Both JAX-WS 2.0 and JAXB 2.0 are being prepared for bundling into post-Java 5 versions of the J2SE specification. Including these components as part of the standard JVM installation will undoubtedly increase their appeal to developers, since it will avoid the need for including these somewhat bulky frameworks in individual applications. The downside of including them in the standard JVM (besides adding to the base download size) is that it may make for versioning difficulties when bug fixes are needed, as we've seen with components such as JAXP, which have gone the bundling route in the past.

Moving toward interoperability

JAX-WS 2.0 directly supports XOP/MTOM, but not the other new WCF technologies. However, as part of Sun's stated commitment to interoperability with Microsoft they have announced they'll be developing open source Java versions of the rest of the technologies included in WCF. These open source implementations will be developed as part of the "GlassFish" megaproject that encompasses all the technologies used as part of Sun's application server (including the JAX-WS 2.0 and JAXB 2.0 reference implementations).

Until these new open source projects take shape, not much can be said. Sun's stated timeframe for the projects is to have something available by mid-year 2006, though, so later articles in this series should be able to supply more details.

The Apache approach

The Apache project has been heavily involved in Web services work for several years, with the main focus on Java platform development. The current production

platform for Java SOAP Web services from Apache is the third-generation Axis framework. Axis is in wide-spread use, both through developers who download and use it directly and from being embedded as the SOAP engine for several different application servers. Axis is generally recognized as the most widely used platform for Java SOAP Web services.

Axis does suffer from some drawbacks, though. For one, it was designed around the JAX-RPC 1.0 standard, which heavily constrained the Axis architecture and limited its flexibility. This limited flexibility became a growing problem as extensions were needed in order to build new technologies around the SOAP processing core. At the same time, the move to doc/lit SOAP services created the need for better schema support than was practical with the Axis code. By mid-2004 the Axis team agreed that a rewrite was needed which would take advantage of the experience gained through implementing Axis, while allowing much greater flexibility for the future.

Axis2 to the rescue

Axis2 is the successor to Axis. It's designed as a light-weight SOAP processing engine (though as with JAX-WS 2.0, Axis2 also includes some support for REST) which is extensible in many different ways. Unlike the original Axis, Axis2 was deliberately not constrained to implement any particular API (though some level of JAX-WS 2.0 support is planned with a wrapper around the Axis2 core code). Axis2 has been in development for over a year, and is approaching production status.

One of the nicest Axis2 features is the AXIOM object model used for SOAP messages. XML object models have been around almost as long as XML itself, starting with the venerable DOM standard from the W3C. What makes AXIOM different from other XML object models is that it uses the flexibility provided by new forms of XML parsers to allow on-demand construction of the object model. This means that you only pay the performance penalty of building an object model for those portions of an XML document which actually need to be accessed through the model.

Another major Axis2 feature is support for pluggable data binding. This feature lets you choose the easiest way of working with the XML payload of your SOAP documents, customizing generated code to use the approach you choose. The possible choices include using AXIOM directly, using a simple data binding approach similar to the original Axis, or using a specialized data binding framework such as XMLBeans, JiBX, or JAXB 2.0.

Extending Axis2

Even though Axis2 is still in active development, there's already a set of projects implementing SOAP extension technologies on top of Axis2. These projects include all the major technologies supported by WCF, along with a few extensions that Microsoft is planning to include in add-on (in other words, separately priced)

applications. Axis2's architecture makes it easy to develop such extensions, using a component called a *module*.

WS-Addressing and WS-Security modules are currently included in the base Axis2 distribution (in the future they may become separate downloads, or even separate projects, since there's no tight coupling between these modules and the core Axis2 code). WS-ReliableMessaging support is being developed through the Sandesha project, while WS-Trust and WS-SecureConversation are being developed through the WSS4J project (which already provides the WS-Security implementation). WS-AtomicTransaction and WS-Coordination are being implemented through the Kandula project.

The minor leagues

Besides the big name choices of Sun and Apache, there are also some innovative Web services projects taking place out in the wilds of open source development. One of these is my own JibxSoap project, a SOAP and REST engine built around my JiBX XML data binding framework. JibxSoap's main strength is its sheer speed -- in early tests, it nearly matched Java RMI performance while using standard SOAP messages. XFire is another SOAP engine, which allows a choice of data binding frameworks to be used; XFire also claims excellent performance results. Both JibxSoap and XFire are likely to reach production status during the next year.

Given the pace of open source development, I wouldn't be surprised to see some other new Java frameworks pop up during 2006. Even if these frameworks don't achieve the popularity of Sun and Apache's offerings, they can still be very influential in showing easier (or faster) ways of accomplishing the same ends.

A look ahead

Now that I've given an introduction to the Java Web services scene for 2006 in this article, I'm going to devote the following articles to covering the open source Java frameworks in more detail. Next month will dive into Axis2, discussing the architecture and the underlying AXIOM object model. I'll also look into the XOP/MTOM attachment support included in AXIOM, and how this is accessed by data binding frameworks. Future articles will cover Axis2 data binding alternatives and performance, as well as the details and performance of other Java Web services frameworks. Check back with each new installment in the series to get the latest details.

Resources

Learn

- Learn more about Microsoft's [Windows Communications Foundation \(WCF\)](#) framework.
- Find out about the [JibxSoap](#) Web services framework built around [JIBX XML data binding](#).
- [Standards roadmap](#) -- understand the impact and importance of standards and specifications for the development of SOA and Web services.
- Browse the [technology bookstore](#) for books on these and other technical topics.
- [developerWorks Java technology zone](#): Find hundreds of articles about every aspect of Java programming.
- [SOA and Web services](#) -- hosts hundreds of informative articles and introductory, intermediate, and advanced tutorials on how to develop Web services applications.

Get products and technologies

- Check the JAX-WS status and get the specification from the [Java Community Process page](#), then go to [java.net](#) to download the latest version of the [reference implementation](#).
- Get the latest [Apache Axis2](#) information and downloads.
- Try out the [XFire](#) Web services framework using your choice of data binding frameworks.
- Get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®. You can [download evaluation versions](#) of the products at no charge, or select the Linux® or Windows® version of developerWorks' [Software Evaluation Kit](#).

Discuss

- [developerWorks blogs](#) -- get involved in the developerWorks community.

About the author

Dennis Sosnoski

Dennis Sosnoski is a consultant and trainer specializing in Java-based [XML and Web services](#). His professional software development experience spans more than 30 years, with the last 10 focused on server-side XML and Java technologies. Dennis is the lead developer of the open source [JiBX XML Data Binding](#) framework and the

associated [JiBX/WS](#) Web services framework, as well as a committer on the [Apache Axis2](#) Web services framework. He was also one of the Expert Group members for the JAX-WS 2.0 and JAXB 2.0 specifications.