Best practices for web services: Part 1, Back to the basics

Formation of a semantic framework

Holt Adams
Dan Gisolfi
James Snell
Raghu Varadan

October 01, 2002

As the hype around web services ebbs and the technology enters the disillusionment phase of the adoption life-cycle, business entities are now demanding best practices to aid them in their technology adoption efforts. This article begins a series that will address the building blocks of web services, applicable business scenarios, and best practice methods for embracing web services by business and IT professionals. Our first task is to go back to basics to lay out a vernacular that will provide clarity to our discussions.

The cloud of technologies being developed to deliver on the promise of web services has (in our opinion) finally reached a point of maturity that allows us to start moving beyond the initial hype and excitement into actual down-and-dirty implementation. In April 2002, Google announced that it would enable developers to query more than 2 billion web documents directly from their own applications using WSDL and SOAP technologies; if nothing else, this demonstrates that the idea of using web services to deliver real-world applications has seriously begun to take root in the minds of developers. Indeed, at many of our customer service engagements, we are witnessing a trend: Customers, and not evangelists, are driving the rapid adoption of web service technologies, within and beyond the enterprise.

A key challenge for many developers and application architects who are seeking to apply these technologies, however, is the fact that amidst all the hype, there is currently very little direction available on how to best apply new technologies such as SOAP, WSDL, and UDDI within real business scenarios. Recognizing this deficiency, a number of developers and architects from across IBM's Global Services, jStart, and Emerging Technologies Groups have sought to identify and document a collection of best practices for the real-world application of web service technologies on a scenario-by-scenario basis.

The process is straightforward. The key to our method is that we will be looking at real business scenarios -- not hypothetical scenarios conjured up to illustrate the way we think things should be,
but situations in which customers are actually applying these technologies within their businesses to solve real problems. Within those scenarios, we ask some very key questions:

- What is the business objective of the solution?
- What general e-business pattern does the solution implement?
- What is the logical architecture of the application?
- How, where, and why were web service technologies used in the building of the application?
- What specific benefit was realized from the use of web services in the application?
- What was learned (positively and negatively) about the technologies and methodologies used?

This article is the first in a series that will document and explore our findings; the series will be most useful to solution architects and others who may be in positions to decide how -- or if -- web service technologies and design methodologies should be adopted for a particular project. Our roadmap for the series will be straightforward. We will first lay a foundation, explaining both the language and the methods we used to identify and categorize the various best practices. This process will include a vernacular, or set of common terms, that we'll define in this first article, plus a second article focusing on the relationship of IBM's e-business patterns (see Resources) to our best practices efforts. The third and subsequent installments will address each of the six questions above for a collection of actual business scenarios. We will conclude with a summation article consisting of our proposal for a set of best practices.

What is a web service?

It is probably fair to say that the initial hype surrounding web services has reached astronomical proportions, so much so that the language used to describe what web services are, and how to go about implementing them, is becoming entirely too overloaded and muddled. In fact, one of the most difficult tasks the recently formed W3C Web Services Architecture Working Group has faced so far has been to determine the answer to what seems like an easy question: What is the generic definition of a web service? Given all the hype, one might think that such a definition wouldn't be hard to come by. The challenge, however, is to overcome the abundance of definitions, most of which are contradictory and only reveal a fragment of what web services can be or may become.

When distilling a collection of best practices, one of the most important first steps is to ensure that the language used to describe the various core concepts is clear, concise, and accurate. Unfortunately, you also have to work within the bounds of already existing and accepted vernacular (regardless of how much confusion an existing name or term might cause). The SOAP protocol itself, for instance, is the poster child for improperly named technologies in the web services world: though the acronym stands for Simple Object Access Protocol, it describes a messaging protocol specification that has little to do with objects and is far from simple to implement.

The working definition of a web service that the W3C Web Services Architecture group managed to agree on is as follows:

A web service is a software application identified by a URI, whose interfaces and binding are capable of being defined, described, and discovered by XML artifacts,
and supports direct interactions with other software applications using XML-based messages via Internet-based protocols.

For those who think that web services are limited to SOAP messages used to invoke the methods of an application over an HTTP connection (as the traditional SOAP stock-quote service does), this definition may be a bit surprising. However, it tells us a couple of interesting things pertinent to our goal of a common vernacular:

- Web services do not require SOAP.
- Web services do not require HTTP.

However, a web service (as defined by the W3C) does require an XML-based description mechanism of some kind (such as WSDL) that can be used to describe the service's form and function. Remember, the backdrop framework for this industry-wide initiative around web service technologies is a service-oriented architecture (SOA; for more information, see the Resources section below). As such, a non-XML based description mechanism (like IDL) would make an SOA implementation more complex and less open.

Note also that the W3C's definition, while mentioning service discovery, does not mention UDDI or any other specific discovery mechanism. This fact will become important as we walk through the various business scenarios and explore how and why certain service discovery mechanisms are used. Specifically, while early literature on the web services architecture asserted that UDDI had a core, essential role to play, real-life implementation of business solutions with web services have demonstrated that UDDI's most significant role is actually quite specialized at this time; in fact, many web services solution are built that do not use UDDI in any way. In the vast majority of current business cases, it would be safe to say that these discovery mechanisms are not yet a core component for integrating processes between business partners.

**Defining our terms, narrowing our focus**

The impact of the W3C's definition of a web service is profound in that the range of things that can be called web services literally includes any software component that can be uniquely identified and described using XML syntax. That range is potentially limitless and does absolutely nothing to help us outline a set of best practices. The term web services itself poorly represents the entire range of applications that may fit into the broad W3C definition, as many such applications may never involve the Web or the technologies upon which the Web is built. Nevertheless, we must somehow include the term web services in our vernacular, as it is already associated with the domain of technologies relevant to our conversation.

What we discovered as we began to analyze customer scenarios is that we needed to come up with new perspectives and a vernacular to differentiate the various subtypes of web services that emerge through the use of the different available description, message, and transport protocols. We started with the Venn diagram illustrated in Figure 1, which identifies a range of possible open protocols that may or may not be used in service-oriented applications.
Figure 1. Service-oriented application protocols

From Figure 1, we derived the first two definitions in our new vernacular:

- **Web services** are software components that are developed using specific technologies from three primary technology categories:
  - An XML-based description format (for example, WSDL)
  - An application messaging protocol (for example, SOAP)
  - A collection or transport protocol (for example, HTTP)

In each of these categories, there are proprietary (vendor- or platform-specific) technologies as well as open (vendor- or platform-independent) technologies available.

- **Service-oriented applications** include applications that MAY make use of web service technologies such as SOAP but MAY NOT include a WSDL or other XML-based description. Such applications are considered *Web-service-like* but are not technically web services.

After considering the W3C’s definition and the Venn diagram in Figure 1, we can say that any piece of software that is described using some XML-based description mechanism qualifies as a web service. It does not matter if such an applications makes use of any other web service technologies. It’s quite possible for an application that uses a programming language-dependent messaging protocol (for example, JMS) and a proprietary transport protocol (for example, MQSeries) to qualify as a fully compliant web service by simply providing a WSDL description of the application’s interface. Conversely, an application that sends SOAP messages over HTTP but does not provide a WSDL description does not qualify as a web service, though it would be considered Web-service-like and deemed a service-oriented application.

So, given our new definition for a web service, let us now focus on the circle in Figure 1 that represents open XML-based descriptions. Furthermore, let's focus on the pain-points at which web service technologies were originally targeted: integration and interoperability. These two pain-points are pertinent to application development both internal and external to the enterprise. Whether you need to integrate with a heritage COBOL application deep inside the back office or with a distributed application on some external server over the Internet, you need to build points of abstraction into your application to ensure ease of integration and interoperability. A *de facto* standard description language in this domain is WSDL, which was created to provide an abstraction layer around a service’s interface and implementation.
Since WSDL is one possible open XML-based description language, there exists a category of web services that choose to use WSDL as their description protocol. In Figure 2, we illustrate such a subset of web services and derive three new additions to our vernacular.

**Figure 2. Domain of web service protocols**

- **Enterprise web services** are web services that do provide a WSDL description but MAY make use of proprietary application messaging or transport protocols. An example of such a service would be one that sends SOAP messages over IBM MQSeries using JMS.
- **Internet web services** are enterprise web services that MUST only use open application messaging or transport protocols. An example of such a service would be one that sends OTA XML messages over HTTP.
- **XML web services** represent the very small subset of Internet web services that MUST use the W3C's adopted XML-based messaging protocol over a narrow range of transport protocols. Specifically, XML web services will only send SOAP messages, and only send them over HTTP, SMTP, or raw TCP/IP connections.

These new definitions capture the concepts conveyed by many in the industry over the last few years whilst providing a semantic framework to categorize the subsets of components that the original (overloaded) term *web services* intended to capture but failed to articulate. For instance, Microsoft's .Net strategy focuses on XML web services, while CommerceQuest's CICS Process Integrator (CPI) focuses on enterprise web services. This new vernacular provides a framework for understanding the intentions of both companies with respect to web service technologies.

**Where do we go from here?**

As we stated at the outset, the goal of this series is to explore a variety of real-world business scenarios and distill from them a collection of best practices for the design and implementation of web services in enterprise environments. Future articles in the series will each focus on a single scenario and will use the language and classifications introduced in this first installment to categorize our findings. The critical element in this process is the fact that this exercise is dealing only with real-world scenarios that very much reflect the point of view of what can be done...
today. We will not discuss theory or hypothetical what-ifs. We will analyze and evaluate each scenario against common, proven design patterns, strategies, and best practices for enterprise development. And we will be completely open about how the use of web service technologies in any given situation helped -- or hindered -- the architecture.

Web services are not a silver bullet, but are a very appropriate alternative for many customer situations. They play a distinct role in the enterprise. It is our intent to provide a set of best practices that will help architects and developers integrate web service technologies into their e-business applications.

© Copyright IBM Corporation 2002
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)