



Web Services

R. Jason Weiss

Development Dimensions International

HTML, ASP, XML... Isn't it funny how so many of the technologies that fueled the expansion of the Web are expressed as inscrutable abbreviations? One of the more recent ideas to charge onto the scene is Web services, which has a plain English name that still manages to be eerily uncommunicative. In this edition of **Leading Edge**, I will try to explain what Web services are all about, how they work, and some of the interesting possibilities they offer for I-O psychology.

XML

XML stands for *eXtensible Markup Language*. XML is typically contrasted with HTML, the original language of the Web. Where HTML tags define how a piece of data should appear (e.g., and to switch bold-faced font formatting on and off), XML tags describe what the data *is* (e.g., <Gender> and </Gender>). The idea behind XML is to permit the development of new languages for describing data using user-defined tags. That's the concept of extensibility: XML document authors create their own tags and hierarchical groupings of tags, called *elements*. The group of hierarchies and tags used to define related information form a language for communicating about that information.

For example, if I were to try creating a simple element to describe simulation stimuli, it might look as follows:

```
<Stimulus>  
  <Name>Delivery Problem</Name>  
  <Exercise>In-basket</Exercise>  
  <Format>E-Mail</Format>  
  <TargetCompetency>Decision Making</TargetCompetency>  
</Stimulus>
```

The HR-XML Consortium, which I have described here before, has groups working to generate flexible, detailed vocabularies for HR-related data such as benefits enrollment, recruiting and staffing, and others (see Weiss, 2001 for more details). One payoff for all of the work that it takes to define vertical industry languages using XML is that they can be shared and used as a common standard for communicating data. Naturally, this benefits companies who

wish to share data. They no longer have to develop a unique data communication protocol with each new partner, but can instead adopt a single, widespread standard and know that any potential partner will also adhere to it. Similarly, companies looking to provide a service can build in compatibility with the industry standard and look to reach the widest market from the outset.

Because it is plain text, XML opens up a world of communication between computers that might be running very different software. When a computer receives XML data over the Web, it doesn't know—or need to know—if the computer sending the data is running UNIX, Windows, or something else entirely. Web services raises the scope of this interchange beyond the mere communication of data to the remote operation of software on other computers, which offers some very intriguing possibilities. Let us now turn our attention to Web services.

Web Services: Software Building Blocks

The simple idea behind Web services is to enable a piece of software to be developed by combining existing software “building blocks” distributed across a wide array of computers, without concern for the compatibility of the underlying systems that provide the building blocks. For example, a company might create a Web site for selecting into a given position by putting together an applicant portal from one computer running Unix, a test administration system from another running Windows, test content from a third running Linux, and an applicant tracking system from a fourth. In some respects, this process is not far from what some companies already do; however, the Web services model standardizes the process.

Let's start by considering the “building blocks” described above. These software building blocks are, in fact, Web services. In its most simple terms, a Web service is software functionality that can be discovered and run remotely (e.g., from a different computer). A Web service can vary in scope, from a service that offers relatively simple functionality, such as a utility that converts dates and times into different languages and regional formats, to extensive and highly complex systems, such as a fully functional applicant tracking system. The important points to remember are that Web services are typically combined as pieces of a bigger, Web-based system.

The value of Web services is that they can be used to tap into existing software functionality, saving the effort and expense of developing it directly. This enables companies to concentrate on developing software where their expertise truly lies. For example, if you are a test publisher, you can focus your development efforts on creating feature-rich software that provides your test content and scoring to other Web services that administer it. You would not need to devote resources to creating a participant portal or any of the other pieces that are necessary in a complete testing system, saving you from supporting software that simply isn't your strong suit. As a consumer of Web

services, a company will be able to quickly put together powerful combinations of Web services to design compelling solutions without having to perform unnecessary development in-house. That's the promise, at least.

How do Web Services Work?

From the above description of Web services as software building blocks, two implications emerge. First, as noted above, a Web service is software functionality that is exposed to the Web and that is invoked remotely. This is to say, software elsewhere on the Web can call this functionality directly, as if it were on the same computer. Therefore, there must be a means for describing and exposing the functionality so that it can be accessed, and there must be a means of accessing. Third, for a Web service to garner users, there must be a means for making it known that it is available.

In the beginning of this article, I noted that Web technologies are an alphabet soup of abbreviations. Web services are actually no different. Aside from XML, three technologies are at the heart of Web services. Web Services Description Language (WSDL) describes how to connect to a Web service and use its functionality. Simple Object Access Protocol (SOAP) is a core communication technology for calling procedures on other computers and delivering the results. Universal Description, Discovery, and Integration (UDDI) are used to make Web services known to potential users. As these technologies are somewhat complex when regarded closely, we will now take just a summary look at each.

WSDL

WSDL documents are like detailed user's manuals for Web services. WSDL documents are written in XML and describe the following:

- What the Web service is;
- Where to access the Web service;
- The types of data required by the Web Service;
- The messages that can be communicated between the Web service and the system calling it; and
- The possible actions that the Web service can be called upon to perform (Worley, 2002).

SOAP

Consider SOAP as the Web services messenger. SOAP is a protocol used to communicate XML. SOAP provides a three-part structure into which XML data are enclosed as follows. The *envelope* defines the message as a SOAP message. The *header* element is an optional component used to communicate peripheral instructions to a server. For example, authentication-related commands would be found in the SOAP header. Finally, the *body* element of the SOAP message contains the XML data that are the heart of the message. These XML data might represent commands for calling a procedure on another system, or the data that are passed in return. The SOAP protocol also provides standards for the com-

position of these types of messages. Finally, the SOAP protocol provides rules for servers to use in dealing with SOAP messages. For example, the SOAP protocol describes when a server should accept or reject data.

UDDI

If you have created a Web service, you will want to publicize it so that potentially interested users will know that it is available. Part of UDDI is the UDDI Business Registry, which can be used to register your Web service so that others can find it or to search for other Web services. UDDI provides several types of directories:

- **White pages** provide contact information for Web services providers;
- **Yellow pages** classify Web services into different categories, much like traditional yellow pages; and
- **Green pages** offer technical details about connecting to and using Web services (Coyle, 2002).

Implications

From the perspective of I-O practitioners who design or deliver software-centric solutions, a future world in which Web services are in full swing offers both tantalizing benefits and significant occasions for pause. Following are some of the obvious implications for I-O of the Web services future.

Focus On Your “Sweet Spot”

As noted above, a Web services architecture lets practitioners devote limited programming resources to the functionality that adds the most unique value. If, say, you have a Web-based job analysis tool, you could potentially add Web services for interview guide generation or survey design and administration. Time and effort are saved, and you don't have a solution that combines the powerful functionality you wish to be known for in your area of expertise with “make-do” functionality in the other areas.

Quicker Assembly of Different Solutions

When the selection of Web services relevant to I-O psychology matures, practitioners will find themselves in the happy position of being able to quickly knit together widely divergent Web services tailored to a client's unique needs. The rapidity of solution development must still be interpreted in terms of software development timelines. However, it makes sense that it should be easier and faster to connect two or more existing systems than to program them from scratch.

Supporting Different Solutions Will Be Complex

The ability to quickly assemble different solutions carries with it the responsibility of having to support the diversity of solutions in play at any given point, each of which may contain different Web services with frustrat-

ingly subtle differences in behavior. Support for these solutions will require an even greater degree of cooperation between the software design and consulting services delivery groups. Another problem associated with the integration of multiple Web services is raised when bugs appear. With a potentially large number of Web services combining to form a single system, locating and eradicating bugs promises to be a frustrating endeavor.

Potential Development of Industry-Wide Standards

Just as HR-XML is working on data communication standards for the world of HR, a future industry consortium might attempt to establish Web services standards for HR. For example, these standards might start by defining the range and scope of different types of Web services within HR and follow with descriptions of minimum necessary functionality required of the different HR Web services.

Final Thoughts

Web services represent a growing movement in the software development community. It remains to be seen whether Web services will ultimately end up producing only reasonably common functionality that can be used in almost any application (e.g., survey administration; help system) or whether intellectual property-heavy, industry-specific Web services will arise. For example, standard vendor-specific competency models might represent an appealing Web service for inclusion in a number of applications. What is most interesting is that there seems to be an attitude of experimentation around Web services in general, as the technology is still fairly new. It will be interesting to see how the I-O practitioner community takes to Web services and more interesting still to see what kinds of applications we are capable of developing 5 years from now.

Questions, Comments, Ideas?

If you have any questions or comments about this or previous editions of **Leading Edge**, or if you have ideas for future columns, please do not hesitate to contact me at jason.weiss@ddiworld.com.

Acknowledgments

I would like to express my appreciation to Fei Chen and Ron Buckton for their helpful clarifications on many of the technical issues described above.

References

Coyle, F. P. (2002). *XML, Web Services, and the Data Revolution*. Boston, MA: Addison-Wesley.

Weiss, R. J. (2001, October). Six things you should know about XML. *TIP*, 39(2), 30–34.

Worley, S. (2002). *Inside ASP.NET*. Indianapolis, IN: New Riders.