Chapter 1

Introduction to Web Services

In an era of computing dominated by the World Wide Web, technology referred to as ***Web services*** stands as a key one for allowing computers to communicate machine to machine, program to program. In the same way that the emergence of the Web transformed the way in which humans communicate with each other and gather information, users of Web services reap great benefits. For example, Web services make it easy to connect all types of computer applications to each other. As you will see throughout this report, Web services deliver a foundation of interoperability greatly needed in a world where computer services and digital information exist in many different forms and flavors.

If, in the future, libraries want to be isolated islands in the ocean of content and information, they can ignore Web services. But because much of what libraries do centers on providing information to library clientele and because information is increasingly more electronic—which causes libraries to overlap with many other organizations in the information sphere—it is necessary for libraries to cooperate and interact with a broad set of other organizations and their technical infrastructures. Web services provide mechanisms that allow libraries to expand their services in many important ways. For instance, they allow libraries to deliver services to patrons through nonlibrary interfaces; enable business-to-business transactions with library suppliers; and support behind-the-scenes search and retrieval in remote resources to enhance service. This report will provide many examples of current and potential capabilities made possible through Web services.

Web services involve a set of international protocols and standards, developed and promoted by organizations such as the World Wide Web Consortium (W3C) and the Organization for the Advancement of Structured Information Standards (OASIS). Unlike library-specific protocols—such as Z39.50, which never gained adoption by other industries—Web services are not library-specific protocols. Instead, they extend libraries' reach to the broadest arenas. Many industries and business sectors have deployed Web services, and companies (i.e., Microsoft, IBM, Oracle, Google, and Amazon) usually include Web services among their strategic technologies.

World Wide Web Consortium (W3C)
www.w3.org

OASIS
www.oasis-open.org

W3C's Definition of Web Services
www.w3.org/TR/ws-arch

In this report, I'll explain why Web services are a set of strategic technologies that libraries need to follow closely and, as opportunities allow, consider adopting. On one level, Web services are a set of technical specifications. More importantly, however, they reflect the reality of a world more interconnected and interdependent than ever before. Organizationally and operationally, libraries increasingly engage in more partnerships, dynamic business relationships, and cooperative efforts, and thus Web services stand as the technology well suited to support organizations engaged in cooperative activities.

Intended Audience

This report aims to provide information on Web services

for a library audience. It will include both conceptual descriptions of the technology and provide some technical information on how Web services are implemented. As the author, I do not assume the reader has any advance technical knowledge and will attempt to explain all the technical terms, abbreviations, and acronyms.

Library administrators or others that need to make decisions regarding library-related technology systems or issues will gain a perspective on the importance of this technology as well as how the implementation of Web services may relate to other library trends and initiatives. Library technical staff will gain from both the conceptual descriptions and the implementation examples.

This report *does not* aim to serve as a programming guide for Web services; however, it should give library programmers a basic idea of the steps involved so they can develop utilities or applications that make use of a Web service or create a Web service. I hope that all readers will gain an appreciation of the importance of this technology, which stands to open up exciting opportunities for libraries.

Basic Concepts

Web services lie within the trend for organizations to follow what is referred to as a ***Service-Oriented Architecture***, which is an organizing principle for an organization's technical infrastructure to support the needs of the organization's software users. Particularly in large organizations, *SOA*, as it is often called, has become the dominant approach for organizing information-technology (IT) infrastructure. In broad terms, SOA involves an orchestrated array of independent software components, each of which provides a well-defined, self-contained unit of functionality. SOA builds on long-standing trends toward distributed computing and delivers the building blocks of a modern framework suitable for a computing age dominated by the Web.

Web services provide the means for implementing this Service-Oriented Architecture. Although, theoretically, other technologies could be employed to construct SOA, Web services stand as the dominant approach and include a set of protocols and standards based on internationally established interoperable protocols, which provide a technology framework consistent with SOA.

Not all Web-service implementations rise to the level of SOA, however. Organizations with complex computing environments may develop strategies that organize their technical infrastructures into modules and components that communicate and operate through Web services. Given the fact that many organizations (with some more able to support Web services than others) work with a variety of software applications, SOA, more likely, is more of a goal than a finished product.

Web services can also be deployed to perform specific functions without necessarily requiring that the entire surrounding infrastructure be based on SOA. An individual set of Web services can be implemented as needed, and, over time, an organization's infrastructure may grow into a more full-fledged SOA.

This report will identify specific XML technologies that comprise Web services, but first, it's important to clarify some common misconceptions.

What Web Services *Are Not*

The expression *Web services* can lead to a bit of confusion, so let's consider a few concepts that might easily be mistaken for Web services, but, in fact, *do not* fall within the scope of Web services.

For example, when discussing Web services, those that are discussing the concept authoritatively are not just talking about systems with a Web interface. Web interfaces deal with the front-end issues of information display and interacting with users through a Web browser, which has become the dominant tool for accessing information and for operating software applications. Not only do most users access basic Web pages through Web browsers, but an increasing number of software applications also rely on a Web interface (rather than graphical applications or windows, such as ones that would be used to work in desktop applications on computers running either a Windows or a Macintosh operating system). With technologies such as AJAX (Asynchronous JavaScript and XML), Web-based interfaces can deliver almost all the user-interface features previously only available through desktop applications. But Web interfaces *are not* Web services.

Web-based information resources are not necessarily Web services either. Just because content might be delivered through the Web does not make it a Web service. Information presented through a Web browser—whether through static Web pages or delivered dynamically from an underlying database or repository—does not constitute a Web service.

Increasingly, computer users receive professional services—such as online banking, travel reservations, map-and-travel-direction services, customer-support services, and online shopping, just to mention a few—through Web-based systems. These Web-based "services" are also not what I mean when I refer to *Web services* in this report.

Additionally, Web services should not be confused with "software as service" or "application service provider." These increasingly popular models for software deployment involve hosting the application on a remote server and providing its users access via the Internet. One of the key advantages of this software model lies in the fact that customers do not have to install and maintain software on local servers or desktop systems. Still, though, software as a service is not the same as Web services.

Finally, many libraries employ a person in the position with the title *Web Services Librarian*. Typically, this position is responsible for the design and maintenance of the library's Web site, which includes the delivery of information resources and Web-based library services. In most cases, however, a Web Services Librarian likely will not be involved with Web-service technologies discussed in this report.

Web Services: Plumbing for the Web

Each of the concepts mentioned previously deal with the front end of the Web, the Web that a user "sees," such as a search-results Google page. Web services, rather, are a behind-the-scenes technology; they provide some of the plumbing needed to connect computer systems. These systems may often have a Web interface, but they don't necessarily have to have one. Web services operate on some of the same lower-level protocols, such as TCP/P and HTTP, associated with the Internet and the World Wide Web.

Not all computer-to-computer communications qualify as Web services either. Although they do have some flexibility, Web services follow a specific architectural framework and involve the use of a particular set of protocols and standards. EDI (electronic data interchange), for example, is a computer-to-computer interaction that follows protocols that are not considered Web services.

The framework of Web services provides some of the basic components for e-commerce. Commercial transactions often require the functionality to extract information from multiple sources, e.g., when a user enters into an e-commerce transaction through a particular organization's Web site. In the course of the transaction, the application on the original server may need to contact other applications, either within its infrastructure or from external sources. In order to complete its work, the application may need to find the other applications that hold the necessary information, send a request, and wait for and receive a response. Multiple request-response sequences may take place, and multiple providers may be involved. All these behind-the-scenes Web-service transactions happen unbeknownst to the user. One of the beauties of Web services lies in their ability to bring the resources of multiple computer systems together to perform complex tasks *without* the intervention of the user.

The W3C, the group responsible for the development of Web services, defines Web services as follows:

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL [Web Services Description Language]). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. (www.w3.org/TR/ws-arch)

One of the key trends in technology involves distributed computing. Rather than create monolithic, self-contained applications, developers often prefer to create multiple smaller systems, each with a specialized function. Each component performs a specialized, well-defined task. It is often easier to develop a group of small, specialized components than a single, complex system. These small components can often be used in multiple applications, saving significant development resources in the long term. Additionally, these small, self-contained components can be considered Web services when they implement the workflows and standards associated with this environment.

Web services can also breathe new life into older software. Legacy systems, often built with tools and technologies now considered outdated, can be adopted to participate in a service-oriented environment.

The Web-services model is designed to be indifferent to how any given computing task is accomplished. A software application can be thought of as "wrapped" by a Web service; in other words, the Web-service layer is not affected by how a particular task is accomplished or what hardware or software does the work. The job of a Web service lies in transmitting a request for a particular task to be accomplished and then delivering back the results.

Although a new application will likely be specifically designed to operate as a Web service, legacy applications can be turned into Web services by simply creating the communications layers that enable the Web-service protocols. By developing Web-service gateways, organizations can help preserve investments that went into developing previous generations of software applications. The internal-database architecture and content and business logic of a legacy system can continue to operate as before, with the application's input and output transmitted through the Web-service layers.

Web services hide the underlying hardware and software from view, and the Web-services model facilitates interoperability, neutralizing differences in hardware platforms, operating systems, and development tools. Once these components are wrapped in the Web-service protocols, all software components appear the same to the users in the "outside world." In addition, Web services can be implemented on all matter of hardware and software platforms—Linux, Unix, Mac, or Windows. Although it's possible to run Web services on desktop systems, most reside on server-class hardware and operating systems.

Web services can operate in a stateless mode, or they can be programmed to operate within specific, defined sessions. Many simple Web services work independently of any particular context or complex series of events. This type of Web service consists of a simple, discrete, request–

response transaction. In some circumstances, however, it is important to maintain a session among the participants in an operation of Web services so that multiple Web-service requests, i.e., those connected in a complex transaction, can be orchestrated.

In a distributed computing environment, no single computer application is likely to be entirely independent. It's often necessary for one computer component to rely on others to obtain specific items of information or to perform calculations or conversions. When a software application needs information to perform its function, Web services provide a mechanism for requesting that information, a way to move that information, and how to structure that information for transit.

Web services provide a layer of communication technology that recognizes the interconnectedness of organizations and their related spheres of expertise, information, and services. It is a technology that allows organizations to better fulfill their roles as suppliers of services and provides opportunities to create value-added services based on enhancing services derived from others.

Web services can be considered a mature and well-established computing model. As a technology, Web services emerged in the late 1990s and have been gaining ground since that time. The standards involved are well defined and broadly adopted, and a plethora of tools exist for developing a Web service. This is not a wait-and-see technology; Web services are currently in the mainstream and can be considered a safe investment for libraries.

Web-Service Components and Protocols

Although a number of technologies might be employed to implement a service-oriented architecture, the technology realm of Web services involves a particular set of standards and protocols. In this section, I'll delve into the acronym and abbreviation soup associated with Web services.

Role and Actions

The basic concepts of Web services focus on the roles and actions that software components play in the overall process:

- A *service provider* is a software component that has some type of function or information resource that's offered for use by others. The service may involve performing a computational task or returning a piece of requested information from a database or repository.
- A *service consumer* is a software application that initiates a request for, and makes use of, a service provider.
- A *service repository* provides descriptions of the services available within a given domain. These repositories will be equipped with some mechanism for service providers to register services.

Operations or actions associated with the Web-services model include:

- *Publish*—Once a Web service has been developed, tested, and activated it is considered *published***.** Part of publishing a Web service includes registration in the appropriate service repository.
- *Find*—In order to complete its work, an application acting as a service consumer will need to *find* the appropriate service provider using a service repository.
- *Bind*—Once the service has been identified, the service consumer will *bind* it, which involves locating its specific location on the network, contacting the service provider, and invoking its service.
- *Service Request/Response*—To invoke a Web service, the service consumer will issue a *service request*. Upon successful completion, the service provider will deliver a *service response*.

Workflows for Web Services

The Web-services model assumes a particular flow of communications among applications. Specific roles and actions apply. (Figure 1, below, illustrates the general flow.) Let's step through the process.

- Step 1 begins with a user making a request. For instance, say a user has accessed a Web page with a panel that displays the local weather. The server that hosts the Web page doesn't have this weather information stored locally; instead, in order to display it, it depends on an external ***published*** service. That Web server, which hosts the local Web page, in this context, becomes a ***service consumer***.
- In step 2, this service consumer issues a ***service request***. (This scenario assumes there's a computer out

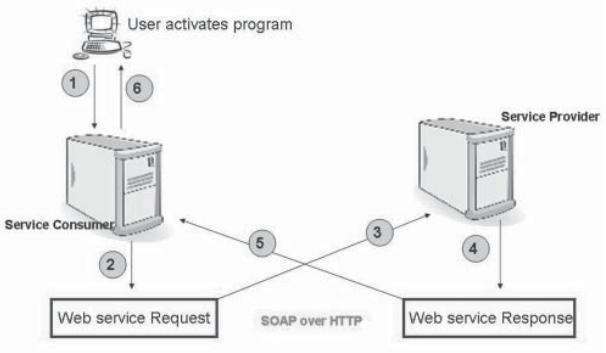

Figure 1:
Communication Path for Web Services

on the Web that hosts a database, or a ***service repository***, of current weather conditions and predictions.)

- In step 3, the Web-service request is packaged as an XML document and delivered over the Internet using SOAP messaging via the HTTP transport protocol (***find***). The remote computer now takes on the role of the ***service provider***.
- In step 4, this service provider accepts the user's Web-service request.
- In step 5, the service queries its database and packages it in XML (***bind***) as a ***service response***.
- In step 6, the original application receives the response and presents its information to the user.

Keep in mind that all these steps happened almost instantly and unbeknownst to the person who invoked the information to be displayed on the Web page. The plumbing analogy is a good one to call upon in order to understand Web services; basically, Web services are just fancy plumbing for the Web.

Although Web services operate as a set of technical mechanisms, they represent agreements and contracts among individuals or organizations. Agreements must be determined in advance—regarding which Web services will be offered by a provider, who may consume the Web services, and at what cost they are offered. Some Web services may be free and publicly available to any requestor, but user authentication and security may need to be added to services that transmit restricted or sensitive content.

In a complex environment, it is often necessary to perform some routing of the message before it reaches the service provider that can fulfill the service request. The final service provider is, therefore, often called the *endpoint*.

The components that comprise Web services are standards developed by the World Wide Web Consortium (W3C)—the body that governs the Web itself. In this context, Web services stand as an important piece of the overall architecture of the Web. The World Wide Web Consortium includes Web services as part of its *architecture domain strategy*.

W3C Architecture Domain/Web Services Activity
www.w3.org/2002/ws

The index page for its Web-service activities is located at www.w3.org/2002/ws. This page links to resources, which describe each of the standards, protocols, and practices operating within the arena of Web services. This Web site provides comprehensive and definitive information regarding the architecture of Web services and the individual components that comprise them. In typical W3C style, the information is presented in a straightforward but terse manner. Those familiar with standards documents will find the content informative, though it makes for difficult reading for those not technically inclined.

Now that I've established a general picture of the workflow, I can move on to identifying the components involved.

XML

XML (eXtensible Markup Language) is one of the most fundamental concepts among Web-based technologies. Almost all applications that deal with data today use some form of XML as the structure and syntax to format data. XML is especially useful when moving information from a software application to another; many applications use XML internally as well. XML provides the syntax for structuring data, and all XML implementations follow the same rules regarding basic syntax. Consistent rules apply to how tags surround data elements, how tags are nested, and how documents begin and end.

Although XML is designed to be processed by computers, it is readable by humans. In previous eras of computing, great efforts were made to store data in the most compact form possible, therefore, data and record formats were often expressed in binary representations that, although compact, could not be easily interpreted by humans. In the library arena, MARC serves as an example of this approach. The leader, tags, fixed fields, variable fields, and sub-field indicators structure bibliographic data in a way that conserves space but is difficult to read.

XML, by contrast, appears quite verbose. Instead of cryptic numerical tags, XML employs descriptive tag names. Today's low-cost digital storage and expanded bandwidth remove many of the previously existing motivations to compress data to formats that only computers can decipher.

Although XML offers great flexibility, it demands exact precision in the way that documents are constructed. Most applications that use XML data verify that the document follows the XML syntax rules. An XML document that passes these rules can be considered *well-formed* XML. Unlike in the realm of HTML, where forgiveness abounds, any error will cause an XML document to be rejected. Yet most Web browsers can perform operations in spite of errors; they are programmed to estimate the author's intentions and then continue to process the page. XML parsers, on the other hand, will reject a document if even a single error is present.

The tags available for use in any given XML document are defined according to the needs of the application, and through agreements, are established among communities of users. The members of an academic discipline, a business sector, or other groups of individuals or organizations that deal with similar types of information have a common interest in following the same XML

conventions. These communities may develop their own DTDs (Document Type Definitions) or XML schemas that define XML tags, data types, vocabularies, and other rules to ensure compatibility of meaning.

In the Web-service arena, great attention must be paid to the namespaces and schema that apply to any given XML document. Because Web services involve exchanging data among diverse participants, it is essential to ensure that the data structures used are compatible.

The code in figure 2 (below) illustrates a very simple XML document. It does not follow any specific schema or DTD, but simply provides a citation using XML syntax. Notice that the document begins with a header that describes the version of XML it uses and the character set, or *encoding*, used for all data within the document. One of the trickiest aspects of XML involves making sure that all data adheres to the encoding rules established. Special characters and punctuation marks often need to be converted to an acceptable form, lest the XML document be invalid and be rejected by the parser.

In figure 2, you can see a number of XML characteristics from this simple example. Each XML document must have a root element—a tag that marks the beginning and end, encompassing all the other tags. In this example, **<citation>** and **</citation>** serve as the document root. Figure 2 also illustrates the fundamental syntax rule that all tags have matching opening and closing elements. Any unclosed tags will cause the document to be rejected.

Those who plan to develop applications based on XML will need to become intimately familiar with all its nuances.

SOAP

As has been explained, Web services involve sending a request from a service consumer to a service provider and receiving a response. SOAP functions as one of the main mechanisms for transmitting the messages—between the service consumer and the service provider—involved in a Web service.

SOAP is a technology developed under the oversight of the W3C; the documents that describe the current version of the protocol reside on the W3C Web site (www.w3.org/TR/soap).

```
<?xml version='1.0' encoding='UTF-8'?>
<citation>
  <author>Breeding, Marshall</author>
  <title>Library Technology Guides</title>
  <type>Web site</type>
  <url>http://www.librarytechnology.org</url>
</citation>
```

Figure 2:
Illustration of a Very Simple XML Document

Previously, the *SOAP* acronym stood for *Simple Object Access Protocol*. But because the protocol really isn't that simple, and it deals with data structures in many ways (as objects as well as many others), the purveyors of the latest version of the standard chose to simply refer to it as SOAP—it is no longer considered an acronym. In its usual terse language, the W3C specification introduces SOAP in this way: "SOAP Version 1.2 (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics."

Fundamentally, SOAP is a protocol for transmitting messages, and of course, those messages will be formulated in XML. In practical terms, SOAP specifies how to construct the XML document that represents the message carrying the request and response. The current version, SOAP 1.2, relies extensively on XML schemas. The crucial role for SOAP lies in wrapping the requests and responses according to the specification. Figure 3 provides an example of a Web-service request created as a SOAP message.

The example illustrates the components that comprise a SOAP message. Because it is in XML, the document begins with the standard declaration and encoding statement (<?xml version='1.0' encoding='UTF-8'?>). Next, you can see that the document's root element is the SOAP envelope (<SOAP-ENV:Envelope). The envelope provides the locations of the schemas that support this instance of SOAP. The SOAP body carries the methods and data that represent the request or response.

Once properly formed or formatted, SOAP messages then can be transported to the proper destination by the Web-service application. SOAP does not specify the method of transport, and although Web services tend to rely on HTTP to transport SOAP messages, other methods can also be used, including a TCP socket, a simple mail message, or a MIME attachment.

WSDL

Web Services Description Language (WSDL) is used to describe a service. The WSDL consists of an XML document that can be used by a service consumer to automatically configure itself to invoke a Web service from a service provider. Many SOAP environments are able to automatically and dynamically configure themselves by simply parsing the WSDL document. WDSL can also serve as documentation to a programmer that wants to construct a static Web-service client.

Not all Web services have corresponding WSDL documents. It is possible to access a Web service based on

advanced knowledge of how it works. If the same developer, for example, creates both the Web service and the client that accesses it, the formality of a WSDL may not be necessary.

For more complex environments that include many services, WSDL is more of a necessity. The WSDL will include seven sets of definitions:

- <type> Specifies the data structures and types involved in the service messages.
- <message> Definitions of all the messages involved. In most cases these messages include the service request and the service response.
- <portType> The definitions related to the interface for each of the messages defined.
- <operation> Describes the operations performed by the services in terms of the messages involved.
- <binding> Specifies the protocols that will be employed to transport the messages and the type of data encoding.
- <service> Provides the URL of the service on its host server.

UDDI

Universal Description, Discovery, and Integration, referred to as the abbreviation *UDDI*, is a protocol that functions to find Web services within a domain. At heart of this process lies a UDDI registry—a repository of all the WSDL XML documents that describe each of the services in the domain. Within a domain serviced by UDDI, part of the process of publishing a service includes registering the WSDL to a UDDI registry.

The UDDI registry plays a role only in helping a service consumer locate a suitable service provider. Once a service consumer receives the location of the service, UDDI binds the service with the service provider directly.

Not all implementations of Web services will have a UDDI registry. In many cases the service consumer will learn about a service via other means. Most public developers of Web services (such as those that developed the popular Amazon Web Service) will discover the services available through documentation provided by the service provider.

Larger-scale implementations of Web services, with a complex matrix of service consumers and service providers, will rely on UDDI to enable applications to operate without human intervention. Domains that have a large number of Web services will find UDDI a useful tool, making the environment more manageable. When a developer makes a change to the service or adds a new service, registering the service in the UDDI will eliminate the need to make programming changes in Web-service clients.

A UDDI registry can be either private or public. A private registry operates within a closed domain, such as within the enterprise network of a company. A public registry is available to all users on the Internet. There was a public registry, *UDDI Business Registry*, which was operated by such companies as IBM, SAP, and Microsoft, but this registry was phased out in early 2006.

SOAP: W3C's Current Protocol Information
www.w3.org/TR/soap

Organization for the Advancement of Structured Information Standards (OASIS)
www.oasis-open.org

The latest version, UDDI v3.0, has recently been ratified as a standard of OASIS (Organization for the Advancement of Structured Information Standards). This version can be considered as completely interoperable among both private and public registries. The interoperability between public and private registries in this version of UDDI obviated the need to maintain the UDDI Business Registry.

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
 <SOAP-ENV:Body>
  <ns1:getLibraryRequest xmlns:ns1="urn:LibrarySearch"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <query xsi:type="xsd:string">Vanderbilt University</query>
    <SearchType xsi:type="xsd:string">LibraryName</SearchType>
  </ns1:getLibraryRequest>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 3:
Web-Service Request as a SOAP Message

UDDI is a technically complex specification. Although the concept of the registry is simple, the API (application programming interface) used to submit documents and make queries is one of the more challenging components to implement.

Network Transport

TCP/IP and HTTP play a supporting role in Web-service interactions. The messages involved in an operation of Web services rely on standard network protocols. Although it is theoretically possible to use other protocols, TCP/IP is ubiquitous and is used by almost all networks that employ Web services. And though TCP/IP might be a given, there are options for the specific method for which a Web service might be implemented. The most common approach involves transporting Web-service messages using HTTP, the same *HyperText Transfer Protocol* used by the Web. Web services that operate over HTTP can take advantage of the same Web-server software—usually Apache—that delivers standard Web pages. It is also possible to create Web services that operate over TCP sockets. Such a Web service would provide its own *daemon* (a program that runs unattended to perform continuous or system-wide functions) that monitors the network for incoming requests.

REST

The model of delivering Web services through SOAP, WSDL, and UDDI can be a complex undertaking. This approach pays off in an environment with a full-fledged service-oriented architecture in place, which supports a complex set of business applications.

There is a simpler approach available, however—the Representational State Transfer, or REST. A Web service based on REST is often called a *RESTful* Web service.

A standard Web page responds with an HTML document; a RESTful Web service involves responses formatted in XML. REST must live within the constraints of the methods available in HTTP: GET, POST, PUT, and DELETE.

Basically, in the REST model, a service consumer sends its service request to a service provider as a URL; then the service response is returned as an XML data stream. It is up to the service consumer to parse the XML stream and make use of the results.

A Web service based on REST can still be described through a WSDL and can be registered with UDDI, though it doesn't have to be. The primary difference for REST involves the absence of the SOAP layer.

Although a RESTful service can often be tested by using a Web browser, in actual operation, the URL that invokes the service is sent by some other software application. Keep in mind that even RESTful Web services are behind-the-scenes operations, and in actual production, Web services based on REST are still computer-to-computer operations.

The advantages of REST include simplicity and speed. Implementing SOAP can be complex, and transferring messages through SOAP involves some overhead, which can reduce performance and response time. REST simply delivers a response as an XML stream, which can deliver information very quickly.

RSS: A RESTful Web Service

A prime example of a Web service in the REST style is *Really Simple Syndication* (RSS). Though RSS finds many uses, the primary appeal lies in its ability to disseminate newly created information—called a *feed*—data that tells interested readers new items have been posted on a blog, news service, or Web site. Many libraries use RSS to disseminate information about library events, new acquisitions, or to deliver search results.

A resource with an RSS feed will usually have an XML or RSS graphic (see figure 4 below). This graphic not only indicates the presence of an RSS feed, but links to the URL that generates the XML document that represents the feed.

RSS is activated through a URL that, when invoked, responds with an XML stream of the items to be distributed. Although it is possible to click the RSS URL and view the XML in a browser, the normal approach is for a RSS reader or aggregator to invoke the RSS link, accept the XML document, and then format and present the RSS entries for the viewer.

The Library Technology Guides Web site includes an RSS feed that you can view as an example of a RESTful Web service. The feed on this site can be invoked with the URL www.librarytechnology.org/rss/rss.pl.

When invoked, the Perl script produces an XML document (see appendix 1), which can be parsed and displayed by a RSS reader.

The Perl script that dynamically generates the XML document for the RSS feed is displayed in appendix 2.

Web-Service Security

The realm of Web services has all the same security concerns as other networked systems. It is essential that a Web service provides information only to the intended recipients and that sensitive information is not exposed in such a way that it can be intercepted by an unauthorized third party. All of the authorization and authentication mechanisms available for general network and Web operations can be applied to Web services. It is common for Web services involv-

Figure 4:
Typical Graphics Used to Indicate an RSS or XML Feed

ing business transactions, which carry sensitive data (e.g., a credit card number), to be transmitted over HTTPS (secure HTTP) rather than be sent over a network as clear text.

Other Web-Service Specifications and Standards

As presented in this report, some of the core protocols related to Web services include XML, SOAP, WSDL, and UDDI. In addition to these fundamentals, a number of other specifications have emerged, which enhance and extend Web services in business transactions and that demand special considerations not adequately addressed in the basic protocols.

Web Services Interoperability Organization
www.ws-i.org

OASIS WS-Security v.1.1
www.oasis-open.org/committees/wss

BPEL4WS, v. 1.1
www-128.ibm.com/developerworks/library/specification/ws-bpel

These second-generation specifications are in an early phase of development and adoption and are not likely to be found in the current generation of library implementations of Web services. The realm of second-generation Web services is complex and prolific; multiple standards and industry bodies are involved, including the W3C, OASIS, and the Web Services Interoperability Organization (www.ws-i.org). Some of these later standards include:

- *WS-Security* provides an extension to SOAP, and it supplies increased security and stronger guarantees on the integrity and confidentiality of the messages involved in a Web service. The specification supplies a method for authenticating each message and is designed to be flexible enough to work with a variety of security models and encryption technologies. WS-Security v1.1 was approved as an OASIS standard in February 2006 and is documented on the Web (www.oasis-open.org/committees/wss).
- *WS-AtomicTransaction/WS-BusinessActivity*—There are multiple specifications that deal with the coordination of transactions. When multiple services comprise a single business-transaction process, it is essential that all complete successfully as a coordinated transaction. If any part of the process fails, each individual service must roll back its work to the original state. Initially, the *WS-Transaction* specification dealt with this layer; more recently, *WS-AtomicTransaction* and *WS-BusinessActivity* provide transaction coordination specifications for two different categories of business transactions, those of short or long duration, carried out as Web services.
- *WS-Coordination* describes another set of protocols related to the way that individual Web services work together to perform a complex task in a business application.
- *WS-ReliableMessaging* specifies a protocol that ensures that messages are delivered reliably between the components of a distributed application, even in the presence of some type of system failure (such as a network interruption).
- *WS-Attachments* describe how binary files can be attached to SOAP messages without causing problems with XML parsers.
- *BPEL4WS,* or Business Process Execution Language for Web Services, is a specification proposed by a consortium of vendors including IBM and Microsoft. According to IBM's Web site on the specification, "BPEL4WS provides a language for the formal specification of business processes and business-interaction protocols. By doing so, it extends the Web Services interaction model and enables it to support business transactions. BPEL4WS defines an interoperable integration model that should facilitate the expansion of automated process integration in both the intra-corporate and the business-to-business spaces." (www-128.ibm.com/developerworks/library/specification/ws-bpel)

As it has evolved, and continues to evolve, the realm of Web services—and the protocols, specifications, and standards I've highlighted thus far in this report are but just a few—has continued and will continue in its ability to accommodate complex business transactions. The nuances of functionality taking place in these specifications each represent important issues in transaction-oriented business and financial systems. Today, most library applications that involve Web services operate only with the core specifications of XML and WSDL, and they may or may not use SOAP. Very few library Web-service applications even make use of UDDI. But as library-automation systems evolve to use Web services in order to integrate with the financial systems of their parent institutions, they may well need to be more aware and knowledgeable of the specifications that relate to complex business transactions.

Common Web-Service Development Environments

Flexibility abounds when it comes to the development of Web services. All of the major programming languages

and development environments provide some method for creating Web services. The following is just a sample of some of the development tools and environments commonly used for building Web services.

Microsoft .NET

Microsoft Corporation has embraced Web services as a key strategic technology, and it has been actively involved in the development of protocols, standards, and specifications related to Web services. The company calls its Web-service strategy *.NET* and appends this name to many of its products and technologies.

Microsoft Visual Studio .NET is an integrated development environment for the creation of software applications and has built-in support for Web services. There are versions of this environment for several different programming languages, including Visual Basic, Visual C++, Visual C#, and J# (an implementation of Java).

Microsoft is working toward the release of its next-generation development framework *Indigo*, which has been specifically designed for the creation of service-oriented business applications.

Java

Java stands as one of the dominant development environments today. Created and supported by Sun, applications created with Java can operate on any computing platform that supports a "Java virtual machine," which is available for virtually all operating systems. One of the major advantages of Java involves program portability; the same program operates on multiple platforms without the need to modify code or recompile.

Sun offers *The Java Web Services Developer Pack* as a free integrated toolkit for the development of XML and applications of Web services.

Most Web-based applications built with Java will operate in conjunction with a servlet container, such as Apache Tomcat.

Apache Axis

The Web-server software *Apache* powers a very large portion of the Web servers in the world. Apache is an open-source implementation created by a worldwide team of developers coordinated by the Apache Software Foundation (www.apache.org).

Microsoft .NET
www.microsoft.com/net/default.mspx

Java.sun.com: The Source for Java Developers
http://java.sun.com

Apache Axis
www.apache.org

Active Perl Available from ActiveState
www.activestate.com

Axis is an implementation of SOAP for Apache Web servers. With Axis in place, developing a Web service becomes an easier task because it takes care of the SOAP messaging layer. Axis operates with Web services programmed in Java. Apache Tomcat, a "servlet container" provides the application environment in which Java programs execute. In most cases, the combination of the Apache Web server, Apache Axis, and Tomcat will work together to provide an environment for SOAP-based Web services based on open-source software.

Apache Axis replaces the older implementation *Apache SOAP*. Axis has many more features, performs better, and fixes many problems that were present in Apache SOAP.

SOAP::Lite

The SOAP::Lite Perl module provides the ability to create Web services with the Perl programming language. Perl is available on almost all of the major computer platforms and is especially popular for writing Web-based applications. Perl is an open-source application and can be used without cost. Perl is widely used on Unix and Linux variations. In the Windows arena, the company ActiveState offers *ActivePerl,* which includes a binary distribution for Windows. Perl with SOAP::Lite will be used for some of the examples in this report.

Appendix 1:
XML Document Produced When RSS Feed for Library Technology Guides Web Site Is Invoked


```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns="http://purl.org/rss/1.0/"
 xmlns:taxo="http://purl.org/rss/1.0/modules/taxonomy/"
 xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns:sy="http://purl.org/rss/1.0/modules/syndication/"
 xmlns:admin="http://webns.net/mvcb/">
<channel rdf:about="http://www.librarytechnology.org">
<title>Library Technology Guides automation update</title>
<link>http://www.librarytechnology.org</link>
<description>Current news from the world of library automation. Includes press releases and other announcements from companies involved in the library automation industry. Compiled by Marshall Breeding, new entries added within 1 day of receipt. http://www.librarytechnology.org.</description>
<dc:language>en-us</dc:language>
<dc:rights>Copyright 2004 Marshall Breeding</dc:rights>
<dc:date>2006-03-10T22:09:18+00:00</dc:date>
<sy:updatePeriod>daily</sy:updatePeriod>
<sy:updateFrequency>1</sy:updateFrequency>
<sy:updateBase>2004-01-01T12:00:00+00:00</sy:updateBase>
<items>
 <rdf:Seq>
   <rdf:li rdf:resource="http://www.librarytechnology.org/ltg-displayarticle.pl?RC=11852" />
   <rdf:li rdf:resource="http://www.librarytechnology.org/ltg-displayarticle.pl?RC=11842" />
   <rdf:li rdf:resource="http://www.librarytechnology.org/ltg-displayarticle.pl?RC=11847" />
   <rdf:li rdf:resource="http://www.librarytechnology.org/ltg-displayarticle.pl?RC=11849" />
   <rdf:li rdf:resource="http://www.librarytechnology.org/ltg-displayarticle.pl?RC=11839" />
 </rdf:Seq>
</items>
</channel>
<item rdf:about="http://www.librarytechnology.org/ltg-displayarticle.pl?RC=11852">
<title>Sno-Isle Library System extends AquaBrowser library search capabilities</title>
<link>http://www.librarytechnology.org/ltg-displayarticle.pl?RC=11852</link>
<description>(March 10, 2006). Sno-Isle Library System in Washington has integrated Web site searching and RSS feeds into its AquaBrowser Library search interface.</description>
<dc:publisher>The Library Corporation</dc:publisher>
</item>
<item rdf:about="http://www.librarytechnology.org/ltg-displayarticle.pl?RC=11842">
<title>ALEPH 500 Version 18 delivers powerful new functionality for patrons and library  staff</title>
<link>http://www.librarytechnology.org/ltg-displayarticle.pl?RC=11842</link>
<description>(March 08, 2006). The Ex Libris Group is pleased to announce the general release of version 18 of the ALEPH 500 integrated library system. With this product release, Ex Libris continues to offer major enhancements reflecting the company&amp;#8217;s user-centric development strategy.</description>
<dc:publisher>Ex Libris</dc:publisher>
</item>
<item rdf:about="http://www.librarytechnology.org/ltg-displayarticle.pl?RC=11847">
<title>SirsiDynix recognized by Alabama Governor for global trade excellence</title>
<link>http://www.librarytechnology.org/ltg-displayarticle.pl?RC=11847</link>
<description>(March 08, 2006). SirsiDynix announced it is one of eight state companies recognized by Alabama Gov. Bob Riley for excellence in exporting their products and services. Riley presented the Governor&amp;#8217;s Trade Excellence Award to SirsiDynix Chief Executive Officer Patrick Sommers and Vice President of Global Alliances Lamar
```

Jackson during a ceremony in the Alabama Capitol here today.</description>
<dc:publisher>SirsiDynix</dc:publisher>
</item>
<item rdf:about="http://www.librarytechnology.org/ltg-displayarticle.pl?RC=11849">
<title>Amsterdam Public Library System goes live with most advanced AquaBrowser Library yet</title>
<link>http://www.librarytechnology.org/ltg-displayarticle.pl?RC=11849</link>
<description>(March 8, 2006). AquaBrowser Library is live at the Amsterdam Public Library. Next to the AquaBrowser catalog search in the library’s holdings you will find Muziekweb, a Dutch music online library offering a treasure trove of information on music; albums, trivia, reference and music websites. You can listen to music clips live right from the search results.</description>
<dc:publisher>Medialab Solutions</dc:publisher>
</item>
<item rdf:about="http://www.librarytechnology.org/ltg-displayarticle.pl?RC=11839">
<title>Endeavor Information Systems and TDNet expand partnership to introduce re-architected reference linking solution for libraries</title>
<link>http://www.librarytechnology.org/ltg-displayarticle.pl?RC=11839</link>
<description>(March 07, 2006). Endeavor Information Systems announced that it has expanded its existing partnership agreement with TDNet, a leading provider of e-resource management solutions, to deliver a new technology offering, Discovery: Resolver, a system which delivers enhanced electronic access to OpenURL and non-OpenURL target resources. Discovery: Resolver, a replacement for Endeavor’s current LinkFinderPlus product, will utilize technology developed by TDNet to provide accurate context-sensitive linking from OpenURL enabled sources to full text sources.</description>
<dc:publisher>Endeavor Information Systems, Inc.</dc:publisher>
</item>
</rdf:RDF>

Appendix 2:
Perl Script That Dynamically Generates XML Document for an RSS Feed

```
require ("DBtextSQL.pl");
$code = "LTG";

$interval=3600 * 24 * 30;
($psec,$pmin,$phour,$pmday,$pmon,$pyear,$pwday,$pyday,$pisdst)=localtime(time-$interval);
$pfullyear = 1900 + $pyear;
$pmon++;
$now = $fullyear.$dash.$mon.$dash.$mday;
$then= $pfullyear.$dash.$pmon.$dash.$pmday;
$SqlStatement = "SELECT RecordNumber, Title, Publisher, Enum, Description FROM bib
    WHERE (SortDate CT \'$then:$now\') AND (Format=\'Press Release\')
    ORDER BY SortDate DESC";
print "Content-type: application/xml\n\n";
&createODBC;
&XMLheader;
&beginRDF;
&ltgrssHeader;
&rdfItemList;
&rdfItemDescriptions;
&endRDF;
&endODBC;
$logfilename    = "bib-$fullyear-$month.log";
open (SEARCHLOG, ">>$logdirectory\\$logfilename");
print SEARCHLOG "$date|$ip|RSS feed request\n";
close (SEARCHLOG);

sub rdfItemList {
  print "<items>\n";
  print " <rdf:Seq>\n";
  my $i = 0;
  &executeSQL("$SqlStatement");
  while ($db->FetchRow()) {
    %data = $db->DataHash();
    print "    <rdf:li rdf:resource=
      \"$server/ltg-displayarticle.pl?RC=$data{'RecordNumber'}\" />\n";
    # last if ($i == 20);
    $i++;
  }
  print " </rdf:Seq>\n";
  print "</items>\n";
  print "</channel>\n";
}

sub rdfItemDescriptions {
  my $i = 0;
  &executeSQL("$SqlStatement");
  while ($db->FetchRow()) {
    %data = $db->DataHash();
    print "<item rdf:about=
      \"$server/ltg-displayarticle.pl?RC=$data{'RecordNumber'}\">\n";
    print "<title>$data{'Title'}</title>\n";
    print "<link>$server/ltg-displayarticle.pl?RC=$data{'RecordNumber'}</link>\n";
    $data{'Description'} =~ s/'/\&\#8217;/g;
    $data{'Description'} =~ s/\'/\&\#8217;/g;
```

```
      $data{'Description'} =~ s/\&rsquo;/\&\#8217;/g;
      print "<description>($data{'Enum'}). $data{'Description'}</description>\n";
      print "<dc:publisher>$data{'Publisher'}</dc:publisher>\n";
      print "</item>\n";
      # last if ($i == 20);
      $i++;
    }
  }


  sub XMLheader {
    print "<?xml version=\"1.0\" encoding=\"ISO-8859-1\" ?>\n";
  }

  sub endXML {
    print "</xml>\n";
  }

  sub beginRDF {
    print "<rdf:RDF\n";
    print " xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns\#\"\n";
    print " xmlns=\"http://purl.org/rss/1.0/\"\n";
    print " xmlns:taxo=\"http://purl.org/rss/1.0/modules/taxonomy/\"\n";
    print " xmlns:dc=\"http://purl.org/dc/elements/1.1/\"\n";
    print " xmlns:sy=\"http://purl.org/rss/1.0/modules/syndication/\"\n";
    print " xmlns:admin=\"http://webns.net/mvcb/\"\n";
    print ">\n";
  }

  sub ltgrssHeader {
    print "<channel rdf:about=\"http://www.librarytechnology.org\">\n";
    print "<title>Library Technology Guides automation update</title>\n";
    print "<link>http://www.librarytechnology.org</link>\n";
     print "<description>Current news from the world of library automation. Includes press releases and other announcments from companies involved in the library automation industry. Compiled by Marshall Breeding, new entries added within 1 day of receipt. http://www.librarytechnology.org.</description>\n";
    print "<dc:language>en-us</dc:language>\n";
    print "<dc:rights>Copyright 2004 Marshall Breeding</dc:rights>\n";
    if (length($mon) ==1) {$mon = '0'.$mon};
    if (length($min) ==1) {$min = '0'.$min};
    if (length($hour) ==1) {$hour = '0'.$hour};
    $dcDate = "$fullyear-$mon-$mday"."T"."$hour:$min:$sec+06:00";
    print "<dc:date>$dcDate</dc:date>\n";
    print "<sy:updatePeriod>daily</sy:updatePeriod>\n";
    print "<sy:updateFrequency>1</sy:updateFrequency>\n";
    print "<sy:updateBase>2004-01-01T12:00:00+06:00</sy:updateBase>\n";
  }

  sub endRDF {
    print "</rdf:RDF>\n";
  }
```