Journal of International Information Management

Volume 13 | Issue 2 Article 4

2004

Web-Services - The Next Evolutionary Stage of E-Business

Santosh S. Venkatraman
Tennessee State University

Web-Services – The Next Evolutionary Stage of E-Business

Santosh S. Venkatraman
Tennessee State University

ABSTRACT

Web-Services are a set of new technologies that promise to take "service-oriented" distributed computing to a whole new level, and eventually take e-business to the next evolutionary stage. Web-Services, in a nutshell, let organizations bridge communication gaps among their information systems, and build new software applications by "stitching" together existing ones. It is capable of integrating applications written in different programming languages, developed by different vendors, and running on different servers with dissimilar operating systems. Web-Services would enable companies to seamlessly connect their information systems and business processes with those of their partners and customers – thus ushering in a new "service oriented" distributed computing architecture. The purpose of this paper is to explore the nature of Web-Services, explain the essential concepts that enable Web-Services, and understand its major benefits and point out its shortcomings.

INTRODUCTION

Information technology has played a very critical role in modern successful organizations. The Internet "hype," however, has lately come under intense pressure due to the failures of many "dot.coms." Organizations are nevertheless quietly embracing (Salkever, 2003; Smith, 2003) the Internet with open arms because it does provide some very tangible benefits at a reasonable cost.

The real benefit from the first phase of the Internet technology concerned email and its associated benefits. The second phase emerged with the concept of the World Wide Web and the ubiquitous web browser. While email resulted in affordable and convenient messaging, the Web allowed for the inexpensive distribution and browsing of "rich" information content. The "static" nature of the Web was soon enhanced with "dynamic" features using database connectivity and scripting (both client-side and server-side). Dynamic, database-driven Web applications were the beginning of the service-oriented Web architecture, which led to the inception of Web-Services. Many technologists (Stencil, 2002) now believe that Web-Services will be the third phase of the Internet, in which companies can use the Web to easily connect their systems and business processes with those of their partners and customers.

Many of the top software vendors such as IBM, Microsoft, SAP, and Oracle are staking their future on Web-Services. SAP, for example, recently introduced a new bundle of products called NetWeaver (Gilbert, 2003), along with Microsoft and IBM, which will become the foundation to many of the company's future products. NetWeaver packages some previously available SAP software, including Web portal and data analysis applications, and application-server software, with new tools for Web-Services-based software development. The three partners will jointly staff support centers to assist customers in using their products, according to SAP. SAP hopes that the links between NetWeaver and Microsoft's and IBM's popular development products will make it easier for programmers to integrate SAP's business-management applications with custom-built software programmed in Java and Microsoft's .Net (Dot-Net) tools. SAP, based in Walldorf, Germany, is among the largest makers of corporate software in the world with an estimated $7.7 billion in sales last year. Microsoft also is heavily betting on Web-Services with its commitment to the .Net environment.

The purpose of this paper is to explore the nature of Web-Services, explain the essential concepts that enable Web-Services, and understand its major benefits and point out its shortcomings. The next section explains the service-oriented software architecture, and defines Web-Services. In section 3, we describe the core Web-Service standards, which makes this extraordinary technology possible. Section 4, then briefly describes a typical architecture and life cycle for Web-Services. In section 5, the business and technical benefits of Web-Services are explained, and a detailed example of how one organization is leveraging this technology is provided. We finally summarize and conclude the paper in Section 6.

WHAT ARE WEB-SERVICES?

The first major information system architecture was based on a centralized mainframe computer environment. With the advent of the PC, there was a gradual shift to client-server architectures and decentralized networked-computing. The Internet is now evolving into a "service-oriented" architecture, in which software sales and distribution can be viewed as a "service" as opposed to a "product." The business aspect of selling "software as a service" involves selling only the specific functionality that is needed currently by the customer, when it is needed (software would be a much more granular product). When the customer's application needs more functionality, it searches the Internet for an appropriate Web-Service, and updates itself automatically. This feature is called "aggregating and integrating," and is one of the basic tenets of Web-Services.

The value that is created by service-oriented architectures is not only the code that is written, but also in the creative stitching together of other people's intellectual property (Ruh, 2003). Vendors can thus customize software by aggregating and integrating software modules via the web on a more granular basis (Kalin, 2002). Web-Services, when fully developed, would be a highly evolved form of this "service oriented architecture" (McManaman, 2003). A service-oriented architecture, in summary, describes a model in which small, loosely coupled pieces of application functionality are published, consumed, and combined with other applications over a network.

Before defining Web-Services, it is beneficial to understand the need for this technology, given the current state of the web architecture. First, is the need for automated browsing of web pages (currently done manually by humans). While it is not cumbersome for consumer end-users to manually visit and take down information from various web sites, it is very expensive and time-consuming for businesses to do so (human interaction with computer). It would be desirable to write applications that can automatically "search" for other suitable web applications to interact with, and record/use information from them. This "automation" is one major goal of Web-Services.

Second, is the issue of data and application integration. Current middleware and EAI (Enterprise Application Integration) tools are somewhat helpful, but it still takes a good dose of human intervention (high cost) to download data, reformat and clean them, so that other applications can use them. This manual "integration" is very expensive, time-consuming, and does not scale very well (Castro-Leon, 2002). Moving and integrating data should be simple and transparent – this is another major goal of Web-Services.

There are many available definitions (Info-Tech, 2001) of Web-Services. Microsoft, for example, states that Web-Services "offer a direct means for applications to interact with other applications. Applications hosted internally, as well as on remote systems, can communicate via the Internet by using XML (eXtensible Markup Language) and SOAP (Simple Object Access Protocol) messages." XML is a industry standard language (similar to HTML) that allows electronic documents to be "self describing," thus making it much easier to share electronic documents among disparate computer systems. IBM, on the other hand, describes Web-Services as "self-describing, self-contained, modular applications that can be mixed and matched with other Web-Services." Finally, Sun Microsystems definition of Web-Services is based on five key characteristics: accessibility via the Web, exposure of an XML interface, ability to be located via a registry, use of XML messages over standard Web protocols, and support of loosely coupled connections between systems.

For the purposes of this paper, we adopt the definition proposed by the Stencil group (Stencil, 2002). Web-Services is a stack of emerging industry standards that define protocols for programmatic communication among a loosely coupled framework of disparate application systems. The industry standards used by Web-Services are XML, HTTP (Hypertext Transfer Protocol), SOAP (Simple Object Access Protocol), WSDL (Web-Services Description Language) and UDDI (Universal Description, Discovery and Integration). These protocols are vital for widespread adoption of Web-Services, and will be described in more detail shortly.

A Web-Service, in summary, is a term to describe any application that "exposes" its functionality to other applications through the use of the aforesaid open standards. An important point to be noted is that Web-Services are not used to build new systems from scratch. Rather, they are tools to use with existing software applications that you want to "stitch" together to create a "new" integrated application. One still needs core business applications (such as financial and accounting systems), core databases, good infrastructure, and rich computing resources - without them, Web-Services aren't going to do much.

THE CORE WEB-SERVICES STANDARDS

The true enablers of Web-Services are the three core standards SOAP, UDDI and WSDL. These comprise the basic capabilities necessary to build the discrete elements of a services-oriented architecture. SOAP, UDDI, and WSDL, are platform-neutral standards, and thus Web-Services are not tied to any specific aspects such as Java 2 Enterprise Edition (J2EE) or .Net (Microsoft). For the many companies that use both J2EE and .Net in various departments, Web-Services will be able to tie these departments together without regard to the development platform that they have chosen. Interoperability among Web-Services is a critical measure of the concept's success.

While both J2EE (which is supported by many vendors), and .Net, a Microsoft-only approach, can be used as the basis for developing Web-Services, Microsoft has an early lead in the quality of developer tools offered for building Web-Services. Though .Net isn't expected to mature for at least a year, developers are excited about the simplicity with which Web-Services can be created using Microsoft's latest beta of the Visual Studio .Net tools. Visual Studio .Net makes it easy to wrap Web-Services around program logic, automatically generate corresponding WSDL, and map XML to the data stream.

However, not all the standards necessary for serious deployment of Web-Services have been defined yet. Standards for authentication and connection management are still in the works. Both proprietary and standard methods exist for authentication and connection management, but to integrate into Web-Services in a platform-neutral manner, such standards should be XML-based. We complete this section by describing the three core standards SOAP, UDDI, and WSDL.

SOAP and XML

SOAP (Simple Object Access Protocol) is the fundamental message-passing protocol that defines how to send data, typically in XML (Extended Markup Language) format, among applications across a network (typically the Internet). SOAP describes how one application dials up a Web-Service and asks it to perform a task and return an answer. SOAP makes it possible to use Web-Services for transactions—for example, checking inventory in real-time and placing an online order. The actual UDDI request and response is in XML; however, both are enclosed in SOAP envelopes.

The SOAP envelope is simply a container for XML data. The envelope helps to package messages as they are sent over the Internet using HTTP. By using HTTP rather than SMTP (or some other transfer protocol), we can both adhere to a standard protocol and expect an immediate response. Essentially, each UDDI request is packaged as a SOAP message and sent via HTTP to the UDDI server as a special HTTP "POST" operation. When the UDDI server is finished fulfilling the request, it creates the response. The response is also packaged as a SOAP message and sent back to the client as an HTTP response.

SOAP is reminiscent of previous message-passing protocols that didn't perform as anticipated. The Common Object Request Broker Architecture (CORBA) and Microsoft's Distributed Component Object Model (DCOM) are the two best-known examples of such protocols. SOAP, however, differs from its predecessors because it is more functional, easier to implement, and better accepted by the majority of software vendors.

UDDI

UDDI (Universal Description, Discovery and Integration) is a set of protocols and APIs (Application Program Interfaces) that is used to "discover" where Web-Services are located. Currently, most organizations developing Web-Services are working on internal systems in which UDDI's discovery role isn't very critical. Some businesses, however, are developing private UDDI registries that can be used by their trusted partners, but eventually UDDI will become a very valuable service on the Internet. When there are thousands of Web-Services available on the Internet, the ability to search for Web-Services on a global scale will become meaningful and indispensable.

UDDI provides a registry of Web-Services and serves their descriptions to UDDI clients. Clients query the UDDI server using SOAP and HTTP. The response contains information about the queried Web-Services. UDDI is a set of protocols and APIs (application program interfaces) that define a registry repository where Web-Services and their associated WSDL descriptions can be catalogued and searched. In the future, businesses or even automated agent software may first search UDDI registries to find possible suppliers and business partners to link up with.

UDDI defines business units and describes basic services. The standard's three sections help users find information about Web-Services:

- *White Pages* describe companies and provide contact information and business identifier numbers;
- *Yellow Pages* list the categories of business served by each company, including geography, industry and product mix; and
- *Green Pages* contain guidelines on how to conduct e-business transactions with each company, including information on business processes and data format.

UDDI is still being developed, and by mid-2003 enhancements to incorporate the following features are anticipated:

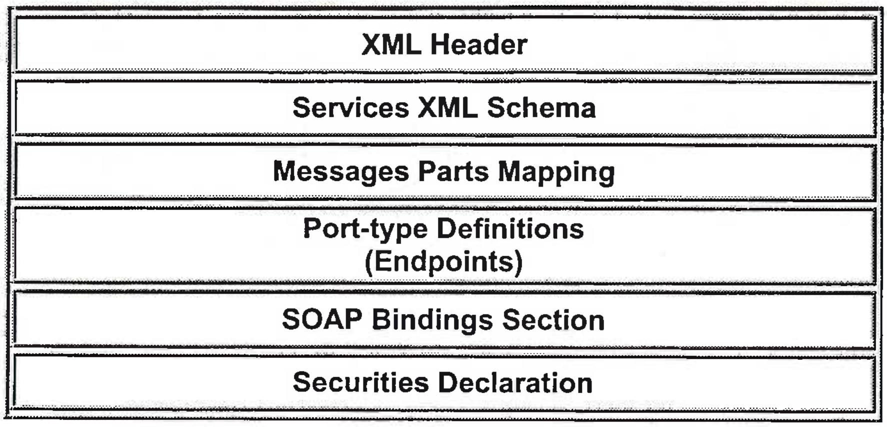
- *Descriptions* of *complex organizations*. Businesses can describe and publish their organizational structure, including business units, departments, divisions and subsidiaries.
- Improved *support* for *international organizations*. Businesses can provide details about products and services in multiple languages.
- Additional *categorization* and *identifier schemes*. Corporations can use industry-specific categories and identifiers to describe their businesses. For example, a chemical company can use existing industry-specific categories to describe itself, its products and its services.
- More *robust searching* options. Companies can search the registry using a wider range of query parameters, more fields and more complex combinations of fields. New options include wild card support and enhanced searching abilities across multiple categories.
- *Stronger business relationship modeling.* This capability allows the modeling of large businesses, complex organizational structures, or various business units and services within UDDI. Modeling business relationships provides the opportunity to make a variety of those relationships—including certifications, alliances and memberships—visible to customers and business partners.
- *Replication capability* that enables the various UDDI Global Business Registries to exchange information more easily. HP, IBM and Microsoft said they expect their registries to support this very soon.
- *Enhance support* for *private trading exchanges.* Improved security is one area of concentration: Companies will be able to track the source of registry information in a registry and whether it has been altered.

WSDL

WSDL (Web-Services Description Language*)* is a standard language that describes a UDDI and specifies how to connect to a Web-Service. With WSDL, service requesters can search for and find the information on services via UDDI, which in turn returns the WSDL reference that can be used to bind to the Web-Service. WSDL is an XML-based language for describing services parameters, attributes, and methods in a standard format, much like a schema for invoking database services. This allows remote application methods to be fed and executed, and results generated and placed in the response side of the transaction request. WSDL, essentially, can be viewed as a declarative binding declaration for distributed application methods.

WSDL-XML documents contain a Header section for namespaces, a Services schema defining element types, a Messages section for element Request and Response mapping, a PortType section that defines endpoints for services, a Binding definition for Input and Output parameters, and a Security declaration section. Figure 1 (McKay, 2002) depicts the schema for the WSDL-XML document structure.

Figure 1. WSML-XML Document Structure



CURRENT STATE OF THE STANDARDS

SOAP, WSDL, and UDDI are platform-independent technologies that make extensive use of XML, a standard language that's used to define protocols and encode the data stream that applications employ to communicate with each other. Though SOAP, WSDL, and UDDI are frequently referred to as "standards," they haven't been blessed by a recognized organization such as the W3C (World Wide Web Consortium) – thus they are just de-facto standards currently. The next major step would be the submission of UDDI, SOAP, and WDSL specifications (if not completely, at least partially) to a credible body such as the Worldwide Web Consortium (W3C).

It is indeed remarkable that the major hardware and software vendors, such as Oracle, HP, Sun Microsystems, IBM, and Microsoft could agree on a set of standards for Web-Services, and state their intention to support and deploy the Web-Services standards in their products. The OSI and other international standards that followed, however, taught us that even when vendors and users agree on a common set of technologies, there could be wide disparities in adoption rates, interpretation of the standards and interoperability among implementations.

Some vendors might try to tweak the standards in their favor, creating incompatibilities and complexity that essentially render the specs useless. If Web-Services are to avoid OSI's fate, vendors must find a way to implement the standards and technology in a common fashion-not only in the laboratory, but in their off-the-shelf products. If even one major vendor disputes the specifications, and does its own proprietary implementation, the resulting confusion could delay the successful deployment of Web-Services for several years.

ARCHITECTURE AND LIFE-CYCLE FOR WEB-SERVICES DEPLOYMENT

Web-Services Architecture

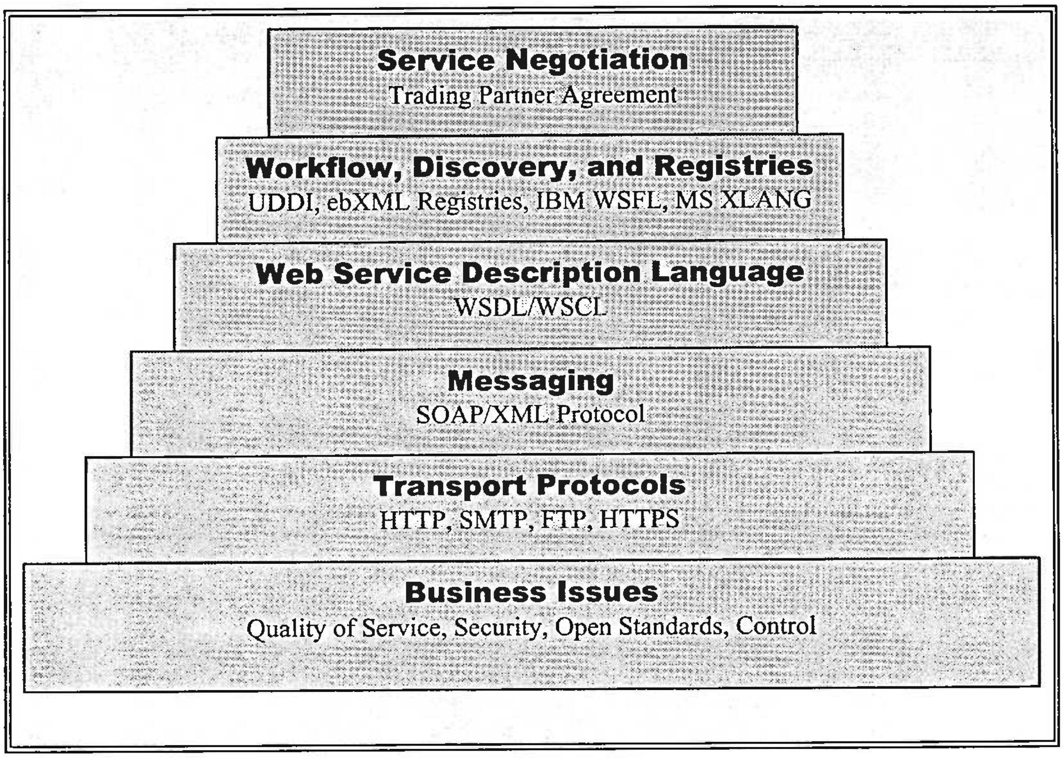
The architecture of a Web-Services stack, including the number and complexity of layers, varies from one defining organization to another. For the purpose of illustration, we have used the schema (Figure 2) of the Web-Services stack as defined by WebServices.org (Myerson, 2002)

Layer 1 is the *Services Negotiation layer* or the *Process Definition layer.* It covers the interactions (documents, workflows, transactions, and process flows) between two or more trading partners that agree on the protocols used to aggregate Web-Services.

The next layer is *Workflow, Discovery, and Registries*, which uses Web-Services Flow Language (WSFL) and MS XLANG, an XML-based language, to describe the creation and function of workflow processes. With

WSFL, you can determine whether the Web-Services should be treated as an activity in one workflow or as a series of activities. While WSFL excels at model presentation, MS XLANG is very good for long-running interactions of composite Web-Services. MS XLANG is implemented in BizTalk, the XML integration server from Microsoft. The second layer also defines Web-Service's transactions with public directories and services. Web-Services that can be made public may get information on credit validation activities from a public directory or registry, such as UDDI.

Figure 2. Webservices.Org's Web-Services Stack


Service Negotiation
Trading Partner Agreement
Workflow, Discovery, and Registries
UDDI, ebXML Registries, IBM WSFL, MS XLANG
Web Service Description Language
WSDL/WSCL
Messaging
SOAP/XML Protocol
Transport Protocols
HTTP, SMTP, FTP, HTTPS
Business Issues
Quality of Service, Security, Open Standards, Control

The third layer is known as the *Web-Service Description Language* (WSDL), which describes the mechanism to connect to a Web-Service. With WSDL, service requesters can search for and find the information on services via UDDI, which in turn returns the WSDL reference that can be used to bind to the Web-Service. Web-Services Contractual Language (WSCL) helps developers use the XML Schema to better describe and develop the structure of data in a more common format.

In the stack's fourth layer, *Messaging*, SOAP acts as the envelope for XML-based messages and covers message packaging, routing, guaranteed delivery, and security. Messages are sent back and forth regarding the status of various Web-Services as the work progresses, say from customer order to shipping products out of the warehouse.

After a series of messages have completed their rounds, they move on to the fifth layer, known as *Transport Protocols,* by using HTTP, Secure HTTP (HTTPS), Reliable HTTP (HTTPR), FTP, or SMTP. Each Web-Service then provides a service requester with services or gives a status to a service provider or broker. Finally, the stack's sixth layer, *Business Issues*, handles other key areas of importance, such as quality of service and security.

The architecture described above is just one of many candidate architectures for Web-Services. Several of the major vendors, such as Hewlett Packard, IBM, BEA Systems, Microsoft, Sun, and Oracle, have their own proprietary architectures for Web-Services. However, for the purpose of this paper, the WebServices.org's architecture described above contains the essential features of the other architectures as well.

WEB-SERVICES LIFE CYCLE

This subsection details the various stages in the Web-Services life cycle (Jenz, 2002), and points out ways in which it significantly differs from the traditional systems development life cycle. The Web-Services life cycle encompasses *design*, *development*, *testing*, *deployment*, *execution*, *management*, and *training*.

Design

CASE tools such as Rational Rose, until very recently, only deal with objects such as components, classes, and interfaces. Now, CASE tool vendors are enhancing their product offerings by providing initial support for Web-Services. Existing classes can be transformed into Web-Services with the help of a software "wizard," which lets the designer specify whether classes should be compiled; deployment descriptors should be generated, etc. From a design point of view, however, a Web-Service is not a new kind of object that needs to be designed. Designers still deal with classes, which are just "transformed" into Web-Services.

Development

A Web-Services Tool Suite provides generators that can create an XML document out of an object and can also transform an XML document into an object. It is relatively easy to create all the necessary instructions to provide a Web-Service "wrapper" to a business service.

When an XML document is transferred as a payload, the receiver needs to validate the document to make sure that it is conformant with its XML schema. Although the sender should have validated the document before sending it, document validation cannot be taken for granted. Although this kind of validation is an extra "overhead," it needs to be considered during the development phase.

Testing

In principle, the functional unit testing of a Web-Service is not different from testing a conventional business service. Each function is executed at least once with a specific set of parameter values. Web-Services can be tested without the user interface, just like server components. If the development tool has generated all the objects necessary to deal with XML documents exchanged between the client and server, no human errors can be introduced, and hence validity checking is not necessary. Testing can, therefore, be highly automated. A test tool can detect interfaces, methods, parameters, and data types on the basis of a set of specified method sequences and generate test cases, which it then executes. Currently, however, there are only few tools to support automated functional testing.

Moving on to load testing requires only a small step. The test tool can set up a specified number of virtual user sessions. Within each session, the tool fires test cases at the service.

Deployment

A deployment descriptor, such as the Apache SOAP Deployment descriptor, describes a Web-Service. It is usually generated by the development tool. Once this has been done, the Web-Service can be deployed to the Web server or Application server. The development tool may be able to deploy a Web-Service automatically.

While Web-Service deployment is necessary for test and execution, it needs to be published only when it is put to use in production. If the Web-Service is confined to internal use within the organization, it is published in the company's local UDDI Registry in the intranet environment.

Execution

A Web-Service can be invoked directly, just like an ordinary business service implementation, if its identity is known. In addition, if the Web-Service has been published to a UDDI Registry, a UDDI client can be used to retrieve the names of existing Web-Services. The user can then select a Web-Service and invoke it from the UDDI client.

Management

Although Web-Services are just "wrappers" around business services; they are manageable entities in their own right, since the Web-Service "wrapper" and the business service implementation are logically tightly linked to each other. In that respect, the Web-Service consists of its language-neutral description and the service implementation. The host environment should provide management functions, such as the suspension and resumption of Web-Services, the removal of Web-Services, and so on. Current products, however, leave ample room for improvement in this functionality.

IT Staff Training and Consulting by Vendor

Currently, getting up to speed with Web-Services technology involves a steep learning curve, but mostly from a "conceptual" point of view. Since XML forms an integral part of Web-Services technology, developers need to get proficient with XML. Developers, however, need to be proficient in XML anyway (not necessarily for just developing Web-Services), since XML is rapidly gaining importance in the software development space.

Excellent tools that offer sophisticated functions to turn existing service implementations into Web-Services already exist - often with just a few mouse clicks. As these tools mature, they will provide even more sophisticated wizards. It will likely be that additional training requirements would be minimal for experienced designers and developers with a solid background in distributed systems.

ADVANTAGES OF WEB-SERVICES

In the previous sections, we covered the basic definition and standards used in Web-Services. In this section, we summarize the major benefits of Web-Services, and illustrate it by describing an actual implementation by Bekins, a trucking company. Web-Services are advantageous for both technical and business reasons, and are summarized in Table 1.

Table 1. Web-Services Advantages

Business Advantages	Technical Advantages
1. Software Management as a Service	1. Lower integration time and cost
2. Dynamic Business Interoperability	2. Rapid integration of legacy systems
3. Cost and Time Efficiencies	3. Less internal human resource needs due to less coding requirements

BUSINESS ADVANTAGES

The modern digital organization often looks to information technology to enhance business aspects. Web-services provide very compelling business advantages as described below.

Software Management as a Service

Unlike traditional packaged software products, Web-Services can be delivered and paid for as "streams of software services," and they permit ubiquitous access from any computer platform. This feature alone is a major advantage as it will change the way organizations sell, purchase, and maintain/upgrade software applications.

Moreover, as Web-Services allow for encapsulation, components can be isolated so that only the business-level services are exposed, while shielding the more technical components from users (Smith, 2003). This results in the decoupling of dynamic components and more stable system components.

Dynamic Business Interoperability

New business partnerships and integrated software applications can be "stitched" together dynamically and automatically on an "as needed" basis, since Web-Services ensure complete interoperability among disparate systems. For example, Milwaukee-based Johnson Controls, which makes climate-control systems, has just announced a new Web-services initiative that uses Microsoft's .Net to integrate systems that have never been tied together before. Johnson Controls is now able to link a school's calendar to its climate-control system so that class room temperatures during school holidays can be programmed from any Web-enabled device (Salkever & Kharif, 2003). New business opportunities thus arise because of this new business model for software sales, construction, delivery, and implementation. Unprecedented flexibility will be afforded to dynamic value chain businesses (LaMonica, 2003; Jain & Juneja, 2003).

Cost and Time Efficiencies

Businesses can be released from the burden of complex, slow and expensive software development and focus instead on value added and mission critical tasks. Web-Services constructed from applications meant for internal use can be easily exposed for external use without changing code. Incremental development using Web-Services is natural and easy; and since Web-Services are declared and implemented in a human readable format there is easier bug detection and fixing. The ease of legacy system integration creates greater agility and flexibility, along with significant time and cost savings. The overall result is risk reduction and more efficient deployment (Welch, 2003).

TECHNICAL ADVANTAGES

Technical advantages refer to those advantages that allow the organization to more easily deploy and manage their information systems. They result in more efficient and effective operation of the information technology department.

Lower Integration Time and Cost

Web-Services are based on universally accepted open "non-proprietary" standards for structured data exchange, messaging, discovery of services, interface description, and business process integration. Software construction and access is via the universally accepted and public Internet system, so software-based business services can be completely decentralized and distributed over the Internet, and accessed by a wide variety of communications devices. Application integration is much faster and cheaper as products are inherently compatible (Taft, 2003).

Rapid Integration of Legacy Systems

Quick integration of "new" tools, as we saw above, is a significant advantage to systems development. However, it has always been a major undertaking, both in terms of money and time, to integrate disparate "legacy" systems with the newer applications. Web-Services, as it turns out, is also excellent for legacy system integration (Kalin, 2002). The state of New Mexico, for example, is creating a Web portal that allows employees to see and tailor their personal information on a single Web page - everything from paychecks to retirement plans. The application uses Web-Services technology in the background to tie together legacy systems, including mainframes, that hadn't been able to communicate before.

Reduced Need For Internal Human Resources Due To Less Coding Requirements

The ability to license and use more software services from third party vendors (Jenz, 2002) results in a significant reduction of "internal" coding, resulting in less demand for application programmers for the organizations. Needless to say, this adds considerably to programmer productivity, and to the organizations bottom-line.

THE BEKINS COMPANY WEB-SERVICES EXAMPLE

An example from the Bekins Company, a Hillside, Illinois-based firm that handles logistics and transportation for manufacturers and retailers, illustrates many of the advantages of Web-Services (Kalin, 2002). Bekins relies on a network of trucking partners to ship tons of big-ticket goods from coast to coast. However, it needed a faster and more reliable way than phoning and faxing, to let its trucking partners know about excess freight sitting on the loading dock, and waiting to be shipped.

Randy Mowen, Bekins' director of data management and e-business architecture, said that Bekins already had a homegrown, mainframe-based scheduling system, written in COBOL; and many of its larger trucking partners already had their own transportation management systems, written in various other languages on different computer platforms. Making Bekins' mainframe-based scheduling system "talk" to its partners' transportation management systems would have taken many application developers several years of custom coding at a high cost using tradition application development. Randy Mowen, therefore, decided to experiment with Web-Services, and develop an Excess Freight Scheduling Web-Service.

In the first phase of the project, Bekins' programmers used a Java programming tool from IBM to write a "tonnage broadcast" application to import information about excess freight out of the mainframe-based freight scheduling system (what the shipment is, where it is, how big it is, where it has to go and what rate Bekins would pay for the shipment) and put it on an IBM application server.

Until now, Bekins' partners could view the excess freight information - smaller partners, through a webpage, and the larger partners, by logging on to Bekins' network. But none of the partners could pull the excess freight information into their own systems, manipulate it and send information back to Bekins. By writing a Web-Service interface to the tonnage broadcast system, Bekins can now send XML messages about the excess freight to its partners. The partners meet Bekins halfway as their software understands the dialect of XML that the Bekins software speaks. The partner systems import the excess freight information from the Bekins system, and sends XML messages back to bid on a given job.

Of the 200 to 300 partners taking advantage of the tonnage broadcast feature when it was first launched in November 2001, about a third were using it as a Web-Service. Encouraged by the results, the next phase of the Web-Services project allowed Bekins to broadcast excess freight information to cell phones and PDAs as well. In summary, Web-Services allowed Bekins to develop a system to talk to any of its partners' systems, and to let those partners talk back to Bekins – all over the Internet. Moreover, Bekins was able to accomplish that in just three months.

While the Bekins example does illustrate the power and flexibility of Web-Services, there are many hurdles in the way of wide scale Web-Services deployment. Current Web-Services standards just can't deliver the same guaranteed, high-level, secure performance that you'd get from traditional business applications. Until they do, Web-Services are not suited for situations where transaction speed, reliability and security are of the essence. Vendors are working on new specifications to address these weaknesses, but it could be another few months before such specifications become standards and get more broadly adopted.

Web-Services are being used by nearly 25% of large corporations in some form or another (Kerstetter, 2002). Merrill Lynch & Co., for instance, has hundreds of Web-service projects under way, including the broadcasting of "electronic alerts" when new research is published. John A. McKinley Jr., the chief technology officer of Merrill Lynch , goes so far as to say that "it's probably the most fundamental technology bet we'll make."

The "Big-Three Automakers" GM, Ford and Daimler-Chrysler, recently announced that they are also placing large bets on the success of Web Services to make them more competitive (Welch, 2003).

The potential reward of Web-Services adoption is significant savings in time and money and a major boost in productivity by replacing many mundane manual processes with automated processes. Web-Services are based on industry standards so that all the services can speak to one another. That keeps companies from having to cope with expensive, proprietary software that can cost up to 10 times as much as Web-services software, and take much longer to develop.

CONCLUSIONS AND FUTURE RESEARCH

Current Web-centric applications significantly expanded the reach and flexibility with which users can access and modify the information locked away in centralized databases. Service-oriented architectures, as exemplified by Web-Services, form the basis for the next evolutionary stage in Web-based application development, in which the application logic is completely decoupled from data in enterprise systems. A Web-Service is a term to describe any application that "exposes" its functionality to other applications through the use of the open standards such as UDDI, SOAP, and WSDL. Web-Services are essentially tools to stitch together existing software applications to create a "new" integrated application.

It has taken unprecedented vendor cooperation and commitment on the design of the core standards (SOAP, WSDL, and UDDI) to make Web-Services happen. In the last 4 years since its inception, Web-Services have not been as widely deployed as anticipated (LaMonica, 2003). Several challenges need to be overcome before Web-Services become a "system of choice" for organizations. These challenges provide abundant opportunities for future Web-Services related research projects such as the ones listed below.

- Identify adoption barriers.
- Identify and benchmark "best practices."
- Develop seamless interfaces for global Web Services providers and subscribers, and comprehensive features to manage public and private catalogs.
- Develop robust security mechanisms for Web Services – both content delivery and peer-to-peer communications.
- Develop scalable architectures and methodology for deploying and developing Web Services.
- Develop metrics to a) evaluate performance and costs of Web Services components, b) calculate Return On Investment, and c) benchmark Web Service flow processes to manage service –level agreements.
- Develop the capability for switching Web Services components in real time (from various candidate sources) to ensure fail-safe transactions.

According to market projections from the International Data Corporation (IDC), the total value of the Web-Services market will reach $21 billion by 2007. Although only 5 percent of companies have completed Web-Services projects in 2002, more than 80 percent are expected to have some type of Web-Services project under way by 2008 (Kawamoto & Ricciuti, 2003]. Web-Services might indeed be the "next big thing" of the Internet revolution.

REFERENCES

Castro-Leon, E. (2002). A Perspective on Web-Services. *WebServices.org*, http://webservices.org/

Gilbert, A. (2003). SAP pushes Web services plan. *CNET News.com*; January 16, 2003, http://news.com.com/

Gunzer, H. (2002). Introduction to Web-Services. *Borland White Paper*, March 2002.

Info-Tech (2001). Web Services: The Business Case. *Info-Tech Research Group White Paper*, http://www.technologynews.net.

Jain, A. & Juneja, N. (2003, July). Business Services Network, The 2nd generation Web. *SOWSIA White Paper*.

Jenz, D.E. (2002, March 18). A view at Total Cost of Ownership and Return on Investment. *Webservices.com*, http://www.webservices.org

Kalin, S. (2002, January). The Essential Guide to Web-Services. *DarwinMag.com*, http://www.darwinmag.com

Kawamoto, D. & Ricciuti, M (2003, February 6). Web Services Market Up For Grabs. *CNET News.com*, http://news.com.com/.

Kerstetter, J.(2002, March 18). The Web at Your Service. *Business Week* E.Biz, http://businessweek.com

LaMonica, M. (2003, January 29). Web services: Bridging the gaps," *ZDnet.com*, http://zdnet.com.com/2100-1106-982525.html

McKay, B. (2002, April) Magical Web Service Tour. *Builder.com*, http://builder.com.com

McManaman, C.(2003, June 7) "Secrets to Designing a Service-Oriented Architecture Using Web Services," *TechRepublic*, http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2914211,00.html

Myerson, J.(2002, June). Examining Two Web-Services Architectures. *Builder.com*, http://builder.com.com

Pallesen, S. (2002, September 9). Why Web-Services Matter. *CRM Forum*, http://www.crm-forum.com

Ruh, B. (2003, July 31). Succeeding at Service Oriented Architecture. *ZDnet.com*, http://zdnet.com.com/2100-1107-5058101.html

Salkever, A. (2003, June 24). How Amazon Opens Up and Cleans Up. *BusinessWeek Online.*

Salkever, A. & Kharif, O. (2003, June 24). Slowly Weaving Web Services Together. *BusinessWeek Online.*

Smith, T. (2003). How Web Services Benefit Wells Fargo, Customers. *InternetWeek*, http://www.internetweek.com/story/showArticle.jhtml?articleID=9901213

Stencil Group (2002, April) "The Laws of Evolution: A Pragmatic Analysis of the Emerging Web-Services Market," *An Analysis Memo from The Stencil Group.*

Taft, D. (2003, January 20). Panel Touts Web Services' Role in Integration. *eWeek.com*, http://www.eweek.com/print_article/0,3668,a=35890,00.asp

Welch, D.(2003, June 24) "Where Web Services Meets the Road," *BusinessWeek Online.*