Doc. ID: ZT-WEBSRV-1
Released: Dec. 10, 2001

zapthink
foundation report

WEB SERVICES TECHNOLOGIES & TRENDS

Q4 2001

ZapThink, LLC • 11 Willow Street, Suite 200 • Waltham, MA 02453 • info@zapthink.com • www.zapthink.com

WEB SERVICES TECHNOLOGIES & TRENDS

December 2001
Analyst: Ronald Schmelzer

Abstract

Web Services is the next evolution in distributed computing using XML as the means by which systems can expose and share computing functionality. The market for Web Services tools, comprised of Web Services Platforms, Application Development and Delivery Suites, and Operations Management is expected to grow to over $15.5 Billion by the end of 2005.

Scope of Report

The main focus of this report is to provide baseline definitions and overviews of key Web Services technologies and offerings by technology vendors. The report aims to also provide key market figures and projections that will help end-users plan for implementation of Web Services in their enterprises.

Key Conclusions

Web Services are Currently More Promise Than Reality

There has been a dramatic shift of attention to Web Services tools and technologies in the past 12 months. However, many of the products, technologies, and standards are still undergoing development. The vast majority of Web Services implementations are in pilot or test phase, and many of the platform products are still in Beta. However, ZapThink expects most of these technologies to be Generally Available by H2 2002.

However, Web Services will Rapidly Dominate as a Computing Technology

Despite the early state of Web Services implementation, the attention by major platform vendors (Microsoft, Sun, IBM, Oracle, Hewlett-Packard, among others) will shape the industry as being Web Service-oriented almost exclusively by 2005. Legacy applications will be made to work within the framework of a Web Services architecture.

Not All Platform Vendors are the Same

There are two major types of Web Service platform vendors: those with a complete IDE, testing, deployment, and application server (service execution) environment and those that only provide components of this solution. While they all provide a platform for development and deployment of Web Services, attention must be given to the features present in the solutions and their ability to withstand increasing competitive pressure from the major platform vendors.

Microsoft .NET and J2EE are Trains Running in Different Directions

There are two major technology infrastructures on top of which Web Services can be deployed: Microsoft's .NET Framework and the J2EE platform. While interoperability is a stated

goal between these two platforms, the set of features, support, capabilities, and strategic direction of these two infrastructures are not in alignment. Users considering adopting a Web Services approach must first decide which "train" they want to follow, and then decide which specific technologies will run within that environment.

Interoperability and Portability Issues Still Remain

Part of the challenge with rapid development of Web Services technologies is that variations in implementation cause interoperability problems. There has anecdotally been a number of interoperability problems between Web Services created on the .NET platform and those created using the Apache SOAP implementation. However, these problems are quickly being ironed out. A greater problem is the fact that true portability of Web Services between platforms is not likely to be possible in the near future.

Application Server Vendors will Rapidly Shift to a Web Services Paradigm

The Application Server market will soon become subsumed within the larger market for Web Services distribution. While Application Server vendors will make the correct claim that Web Services are just one aspect of Web application delivery (including component, scripting, and other traditional deployments), the increasing proliferation of Web Service-based applications will make Application Servers the de-facto environment for Service Execution.

Internally-focused Web Services Implementations is Where the ROI is Now

The best ROI from Web Services can be realized by applying the technology to solve internal application functionality challenges such as EAI and data integration, content management, and rapid application development.

Web Services – Baseline Definition

> *Web Services are application functionality residing on systems that accept requests from other systems locally or across the Internet by means of lightweight, vendor-neutral communications technologies. In the context of this discussion and as increasingly accepted by the software industry as a whole, Web Services utilize XML-based technologies as the communications format.*

Features of Web Services include:

- Loose coupling between application requests and implementations
- Universally publishable, accessible, open, and standards-based service interfaces
- Use of Internet standards and XML for program-to-program messaging
- Mechanisms for service description, discovery, and registry

What makes Web Services different from previous attempts to solve distributed computing challenges are the technologies which provide the service – namely its reliance on the Internet and XML technologies to accomplish those goals.

Some have attempted to retroactively define Web Services as covering all applications delivered via the Internet regardless of its underlying architecture. In ZapThink's opinion, this is incorrect usage of the Web Services terminology. As defined below, Web Services form a distinct class of web-accessible application functionality. Unfortunately, the choice of the term "Web Services" uses two very generic and broadly used words to describe this functionality, and thus leaves much to be desired. Regardless, the term has become accepted by the majority of industry participants as referring to the specific meaning as described above.

For the purposes of this discussion, Web Services will be limited to those that are exchanged using SOAP. Other forms of Web Services, such as those using XML-RPC and other XML and non-XML data formats are not covered here, but many of the approaches can be applied to those technologies as are relevant.

Evolution Beyond Current Distributed Computing Methods

Web Services are the next evolutionary step in Distributed Computing: The major "steps" in distributed computing are, at the highest level:

- First Step: Client / Server technology
  - Thick client protocols such as IIOP
  - Platform-dependent component technology such as DCOM and CORBA
- Second Step: Web Applications
  - Access functionality through a web browser and server
  - Lightweight HTTP protocol
  - Application invocation through HTTP requests
- Third Step: XML-based Web Services
  - Combine lightweight HTTP protocol with structure of Client/Server technology
  - XML-based technologies for data exchange

Tight Coupling vs. Loose Coupling

The primary feature of Web Service implementations is that they are loosely coupled, meaning that implementations can be changed at either end of the connection without breaking application functionality. Traditional distributed computing technologies, such as mentioned above, have some amount of tight coupling requiring assumptions to be made about technical architecture (operating system, object model, or programming language). Other coupling comparisons that are frequently associated with Web Services are illustrated below:

Tight coupling	Loose Coupling
Static coupling	Dynamic coupling
High-cost change	Low-cost change

Message-oriented Nature of Web Services

Web Services are message-oriented in that they wrap up the fundamental units of communication into self-describing packages. This is contrasted against distributed object systems in which the sender must make many assumptions about the recipient (application activation and tear down, interfaces, etc.). On the other hand, messaging systems, including Web Services, form a contract at the wire level requiring recipients only to understand the message being sent.

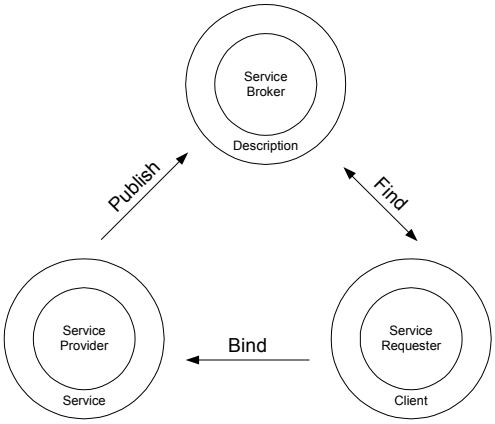
Service-Oriented Architecture (SOA)

Service-Oriented Architecture (SOA) forms a conceptual basis for implementation of Web Services and is comprised of:

- A Service Provider that *publishes* available Web Services to a Service Broker and provides a service interface for Service Requesters
- A Service Requester that *finds* available Web Services at a Service Broker and *binds* to a specific Service at a specific Service Provider
- A Service Broker that provides service registration and discovery capabilities

In an SOA, existing non-service application functionality, such as existing COM and J2EE applications can be integrated and exposed as Web Services as needed. In particular, various additions to these object platforms have actually simplified the process of moving to a more service-centric computing model. In particular, the Model-View-Controller (MVC) architecture in J2EE and the development of the COM+ platform by Microsoft have aided in moving towards a more service-oriented architecture.

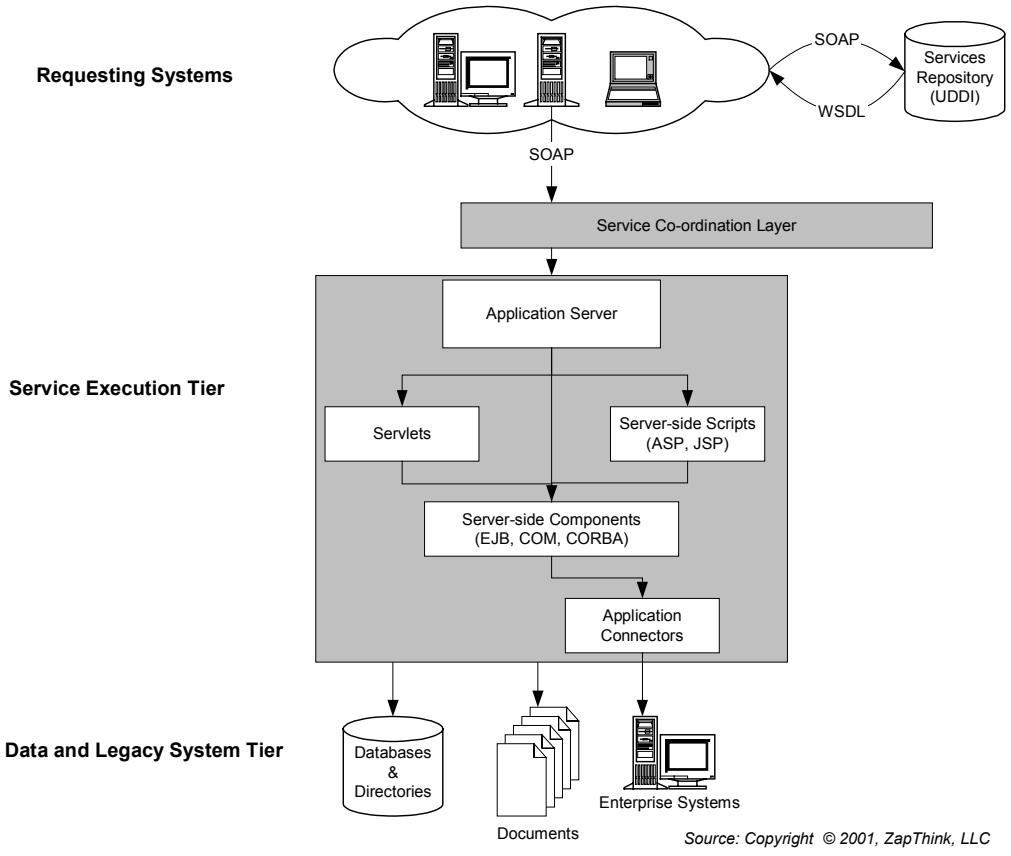
Service-Oriented Architecture



Source: Copyright © 2001, ZapThink, LLC

Location in Enterprise Architectures

Web Services System Interaction



Source: Copyright © 2001, ZapThink, LLC

Remote Procedure Call (RPC) versus Document-Oriented Processing

Web Services understand the context of inbound requests and deliver dynamic requests based on specific application inputs. Since Web Services are based on XML standards, all requests received are XML documents that have to be pre-processed, and optionally transformed to a format understood internally. Security checks need to be implemented before processing the requests. After processing the request, the system must generate an XML document as a response.

There are two basic approaches to inter-system communication:

- Procedure-Oriented (RPC-like)
  - Uses method invocation
  - Proven by technologies such as DCE, CORBA, COM, and EJB
- Document-oriented (EDI-like)
  - Uses document sharing and manipulation
  - Associated with Message-oriented middleware (MOM) such as IBM's MQ Series, Java Messaging Service (JMS), and Microsoft MSMQ.

Traditional RPC services have faced implementation challenges due to their requirements for similar underlying architecture, byte formats, and other technicalities. These challenges spurred development of object request brokers (ORBs) and an Internet Inter-ORB Protocol (IIOP) whose practical and widespread application remains elusive. All varieties of RPC are fairly complex, involving the mapping and reverse mapping of data types and the marshalling of arguments.

Although CORBA and Microsoft's DCOM have been implemented on various platforms, the reality is that any solution built on these protocols has been historically vendor-dependent. Out-of-the-box DCOM and IIOP interoperability is mostly a promise rather than a reality, and have severe limitations in facilitating client-server connectivity when machines are scattered across the Internet. Since CORBA 2.0's launch back in 1995, many IIOP/HTTP gateways were produced to enable IIOP tunneling over HTTP, but this has provided a limited solution to the above problems.

Web Services solves some of these problems by:

- **Using HTTP** to cross firewall boundaries and be payload agnostic
- Employing **XML as an encoding schema**
- Being a **lower cost** framework
- Using **URLs for object identification**
- Working aggressively to **solve interoperability** issues
- Having widespread **industry support**
- Using **late binding** to enable links between systems to be resolved at run-time
- Providing **Dynamic Inspection** so that the availability and functionality of Web services can be discovered at run-time, rather than design time

Problems addressed by Web Services and ROI

The primary goal of Web Services is to simplify inter-system integration and communication. As a result, a number of key problem areas are addressed:

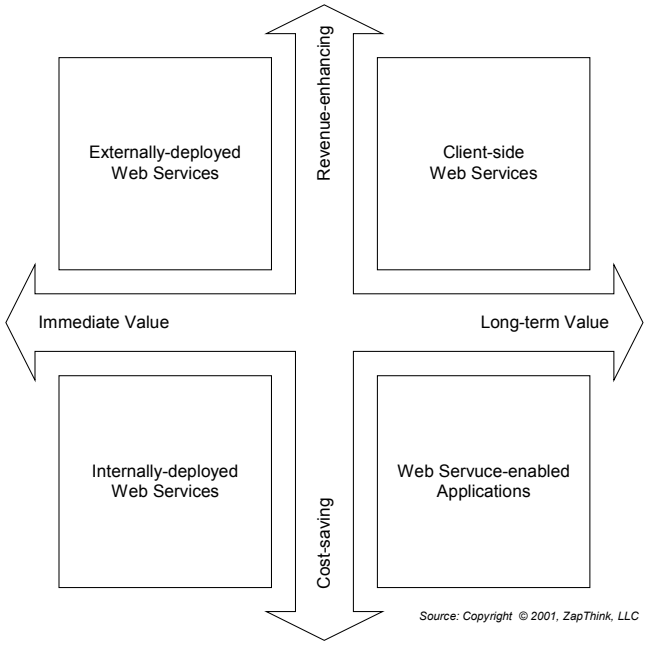
- Enterprise Application Integration and Interoperability
  - "Domain bridge" between COM / CORBA systems
  - Common format for data mapping and transformation
- B2B Integration
  - Cross-firewall distribution
  - Abstraction with business implementation
- Content Management

- Representation of content as discrete services

The primary value propositions of Web Services are:

- **Reduction of Cost**: Web Service applications are easier to integrate, utilize lower-cost tools, and require less time to create
- **Improvement in Efficiency**: Web Services promise a high degree of reusability, faster time-to-market, and ability to integrate with third-party systems and trusted business parties
- **Better Business Operations**: Web Services enable a common architecture and approach to be pervasive in an enterprise containing heterogeneous legacy systems
- **Potential for New Revenue Streams**: Use of externally-supplied Web Services allow maintenance of existing customers or addition of new revenue streams

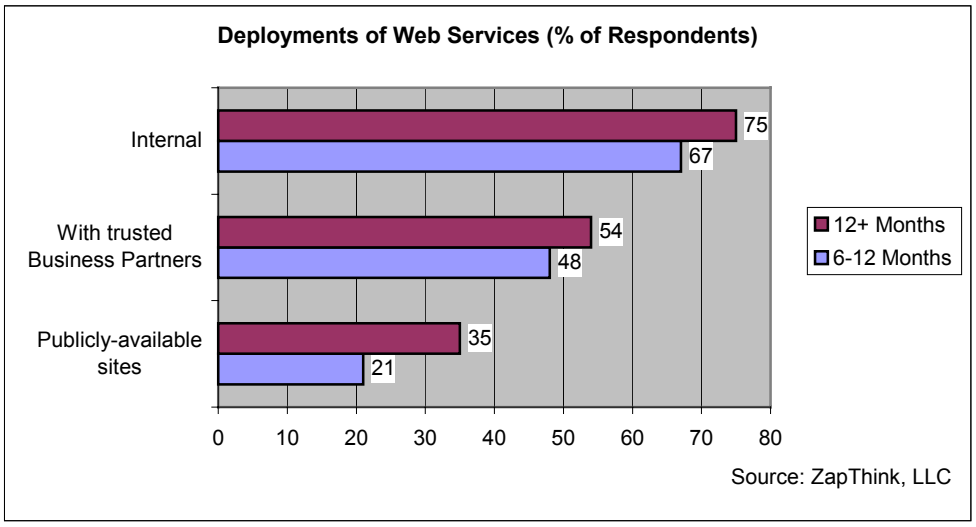
Web Service Implementation Quadrants

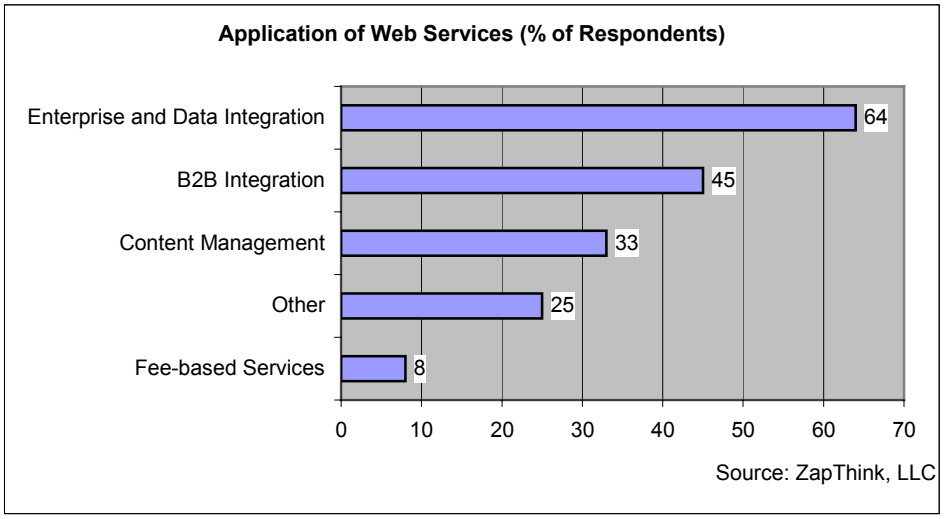


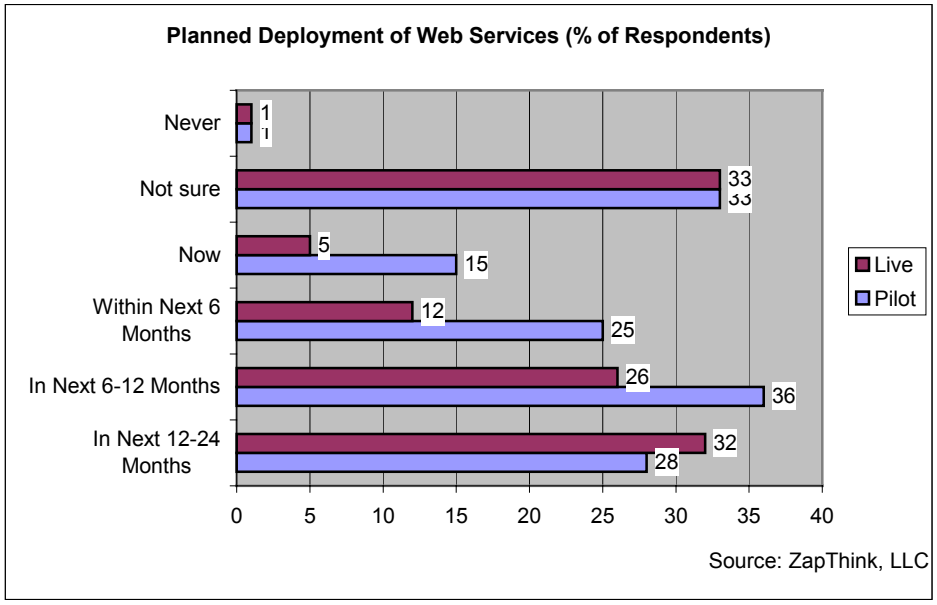
While Web Services are mainly a distributed computing paradigm, there are at least four major ways in which the technology is being employed:

- **Internal, "behind-the-firewall"** implementations focused on enterprise integration, content management, and other high-value, internally-focused applications
- **Externally-deployed** Web Services aimed at exposing business processes to third-parties or trusted business partners and improving customer satisfaction, efficiency, or revenue.
- **Web Service-enabled applications** that make use of Web Services within the confines of existing desktop or client/server applications
- **Client-side Web Services** that use client-side code crafted around Web Service functionality and offer rich user experiences outside the confines of traditional web-based deployment.

As evidenced by early implementation success, the most immediate value for Web Service implementations are in internally focused Web Services aimed at solving integration challenges. Currently, most Web Service applications are extensions of existing functionality and logic, but the long-term strategic value is in natively Web Service-conversant applications that can provide a rich user experience while simultaneously providing the architectural value of Web Services.





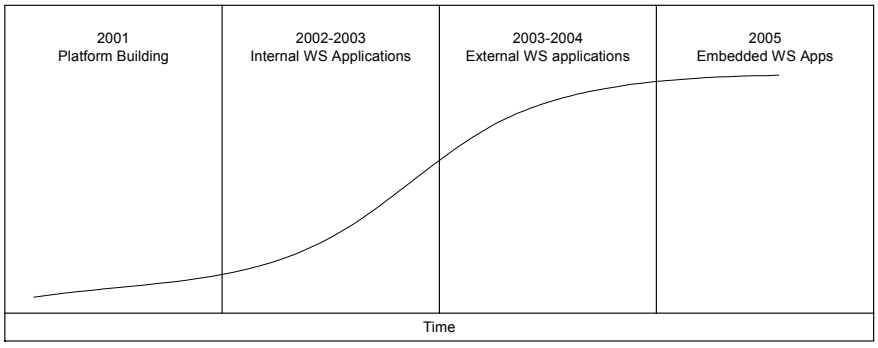


Methodology disclosure: ZapThink conducted survey with sampling of Fortune 500 firms via email, phone, and at events

Current State of the Market of Web Services

Enterprise companies have picked up on the value of Web services for internal application integration. The first phase of Web Service implementation is the establishment of Web Services platforms. As these platforms iron out their kinks, usage of Web Services will proliferate within the enterprise followed by the enablement of external business processes through Web Services. Finally, Web Services will have proliferated enough internal and external business processes that they can then become embedded within shrink-wrapped business applications.

Adoption of Web Services Timeline



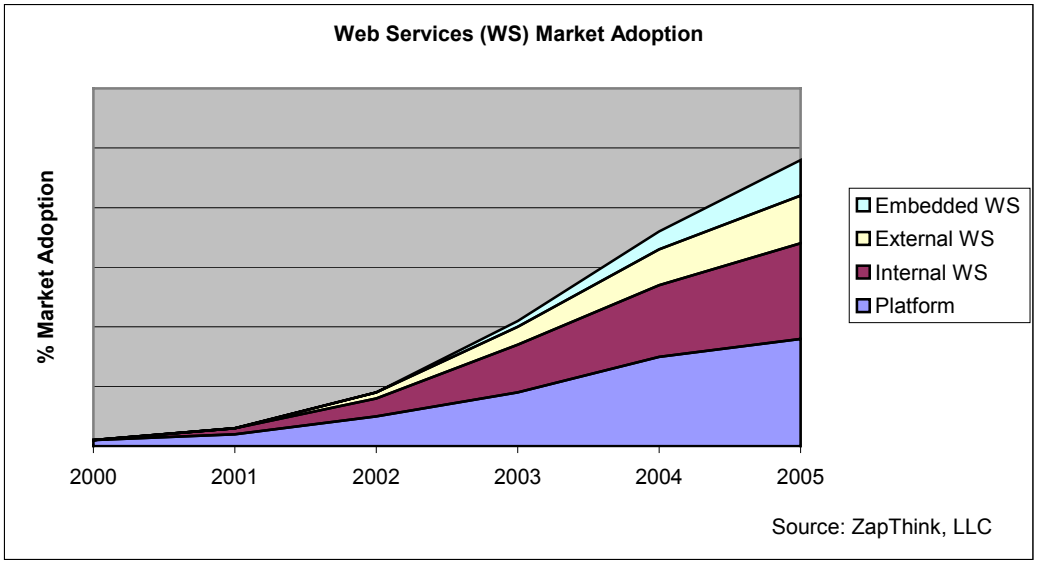


Table 1

Adoption of Web Services Timeline (% of maximum market)

	2000	2001	2002	2003	2004	2005
Platform	5%	10%	25%	45%	75%	90%
Internal WS	0%	5%	15%	40%	60%	80%
External WS	0%	0%	5%	15%	30%	40%
Embedded WS	0%	0%	0%	5%	15%	30%

Sizing & Growth of Market

Based on the above segmentation of the market, ZapThink shows that:

- The Web Services Platforms market will grow from a $320M market in 2001 to a $7.9 Billion market in 2005
- The Web Services Application Delivery market will grow from a $100M market in 2001 to a $5B market in 2005. This includes the Portal Server market.
- The Web Services Operations Management market will grow from a $30M market in 2001 to a $2.5B market in 2005
- The aggregate market for Web Services will grow from a $380M market in 2001 to a $15.5B market in 2005. This includes the Application Server market ($7B in 2005)

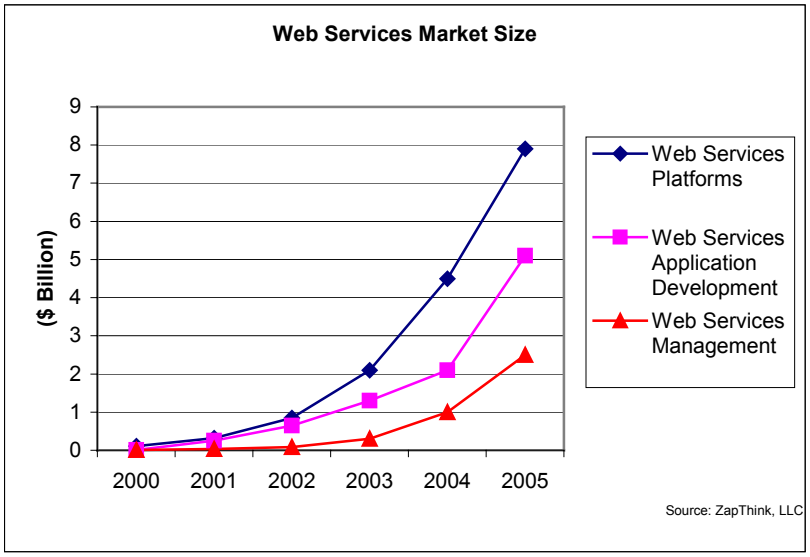
Table 2

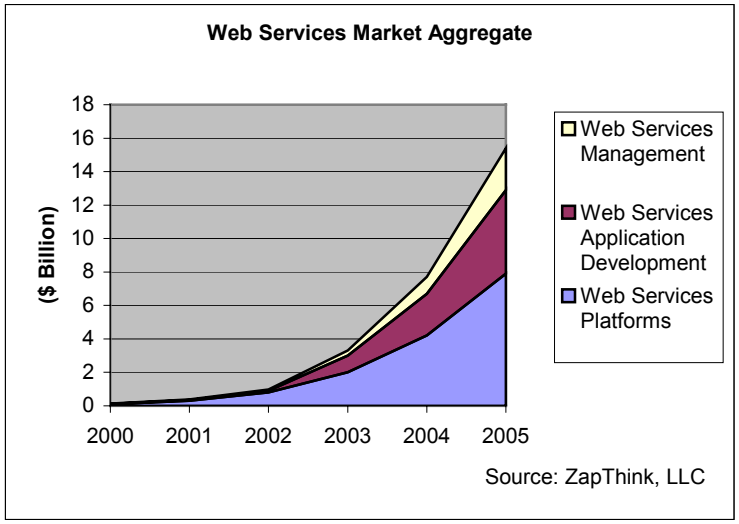
WEB SERVICES MARKET SIZE ($B)

	2000	2001	2002	2003	2004	2005
Web Services Platforms	0.11	0.32	0.85	2.1	4.5	7.9
Web Services Application Development	0.1	0.25	0.65	1.3	2.1	5.1
Web Services Management	0.01	0.03	0.08	0.3	1	2.5

Submarket Segment Sizing ($B)

	2000	2001	2002	2003	2004	2005
Web Services Platforms	0.1	0.3	0.8	2.0	4.2	7.4
Reliable Delivery	0.01	0.02	0.05	0.1	0.3	0.5
Web Services Application Delivery Suites	0.01	0.03	0.2	0.6	1.0	2.2
Web Services-enabled Portals	0.1	0.2	0.4	0.7	1.1	1.5
Web Services Client-oriented Delivery	0.01	0.02	0.05	0.08	0.1	0.3
Web Services Communities	0.01	0.02	0.04	0.08	0.1	0.2





What's Real and What's Not?

In the process of preparing this report, ZapThink has had the difficult task of sorting out what aspects of the technology are available today and what is "promise". A number of claims have been made by a number of vendors (and not just the small ones) about technology that is in various states of alpha, beta, general availability, or even just "marketecture" (marketing ware). As a reader of this report, you have to evaluate two things:

- What state is the solution in as far as release cycle?
- How important is the current release versus future releases?
- What is the schedule for development?
- How quickly are other (competitive) vendors progressing in their implementations?

As an analyst group, it is impossible for us to state the current status of most of the vendors in this report as our report will be read by our subscribers over the next six months – a period of time when products in alpha often become generally available. However, it has been noted that a number of major platform vendors are in alpha and marketing phases for their products. One should be careful in ascertaining the exact status and nature of support for the products being offered.

To evaluate the current releases, ask the following of the platform vendors:

- What products are GA (Generally Available) today?
- What is your product lifecycle and release schedule like?
- What do you know about vendor X's (competitor) release history?

At the current point in time, only about 25% of the profiled vendors have GA product, but we expect most products by all the vendors profiled to be generally available by the first half of 2002.

Pieces of the Puzzle: Standards and Specifications

The standards-based approach of Web Services and XML allows developers to build systems that maximize interoperability and facilitate application development. ZapThink sees the following standards and specifications as key influencers in the Web Services space:

Transport

HTTP	Hypertext Transfer Protocol. Along with its secure variant, HTTPS, the main protocol currently used for most web transactions.
SMTP	Simple Mail Transfer Protocol. The protocol for email transport, which besides HTTP, is a means for exchange of XML documents

Service Messaging

SOAP	Simple Object Access Protocol. Provides a XML-based, wire-level, reliable messaging protocol that supports message-passing and RPC call semantics.
ebXML	e-Business XML. An OASIS-led effort for providing business-to-business service interaction though an end-to-end stack of protocols and specifications for conducting electronic business over the Internet using XML and other open standard technologies.

Service Description

WSDL	Web Services Description Language. Provides an XML schema for description of Web Services in a manner analogous to CORBA/COM IDL. WSDL defines message types, operations on messages, operations, interfaces, bindings, ports, services, and other relevant information for a call to the web service.
WS-Inspection	Web Service Inspection is an IBM-led effort for distributed service discovery method that specifies how to inspect a web site for available Web services by providing references to service descriptions at the service provider's point-of-offering.

Registry

UDDI	Universal Description, Discovery, and Integration. Provides Web Service registration and discovery through a global, public directory of businesses and services that provides pointers to WSDL documents and interfaced via SOAP.

Security

SAML	Security and Authorization Markup Language. Authentication and authorization for Web Services
XKMS	XML Key Management Services. Provides a trusted service to manage public and private encryption keys and distributes them to clients when required.

Transaction

XAML	Transaction Authority Markup Language. Transactional control for multiple Web Service execution. Mostly dead, most participants are working on BTP now (see below).
BTP	Business Transaction Protocol. Specification for working with long-lasting workflows, business processes, and transactions that span multiple enterprises.

Workflow & Business Process

WSFL	Web Services Flow Language. IBM initiative that provides composition and choreography of web services including a core model for workflow focusing on basic process and simple directed edges that control the flow of processing logic from one activity to the next.
XLANG	Microsoft proposal for workflow that handles secure transactions spanning hours or days in an open environment, tracks the state of protocol instances, and detects protocol errors in message flows.
BPML	Business Process Modeling Language. Specification for modeling business processes using an abstracted execution model for collaborative & transactional business processes based on the concept of a transactional finite-state machine.

End-point Definition

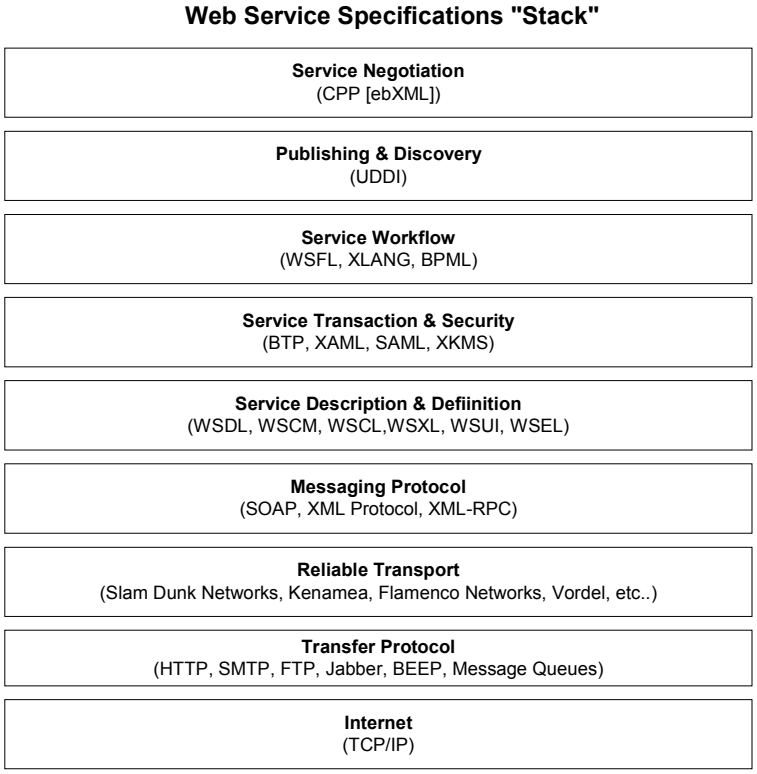
WSEL	Web Services End-point IBM-created language that defines Quality of Service (QoS) characteristics, sequencing of operations, cost, and security characteristics.

User Interface

WSUI	Web Services User Interface (WSUI). An Epicentric-led effort to define an Interface and interactivity specification layer on top of Web Services.
WSXL	Web Services Experience Language. An IBM-led effort to provide a component model for interactive Web applications that enables businesses to distribute Web applications through multiple revenue channels, and enable new services or applications to be created by leveraging existing applications across the Web. WSXL defines three basic Web service types for data, presentation, and control
WSCM	Web Services Component Model. An OASIS initiative for composition and presentation of Web services using a component-based model. WSUI and WSXL are contributing efforts.

Java APIs for XML (JAX)

JAX/RPC	Provides a common API for sending and receiving method calls including marshalling and unmarshalling with SOAP and other XML-based messages while isolating users from protocol specifics. Promises to unify the many different APIs from different SOAP implementation vendors.
JAXM	Provides a standard way to send XML documents over the Internet from the Java platform based on the SOAP 1.1 and SOAP with Attachments specifications. JAXM can be extended to work with higher level messaging protocols such as ebXML. The difference between JAXM and JAX/RPC is that JAXM is message-oriented while JAX/RPC is designed specifically for RPC behavior.
JAXR	An API that helps perform registry activities such as publishing, editing, and querying.
JAXB	Java API that provides a two-way mapping between XML documents and Java objects.
JAXP	Processes XML documents using various parsers through a "pluggability layer", which allows users to plug in an implementation of the SAX or DOM APIs as well as XSLT processors.



Source: Copyright © 2001, ZapThink, LLC

SOAP Implementations

In order to provide the key aspects of Web Services functionality, an execution environment needs to implement the SOAP protocol in a manner that is interoperable with other SOAP implementations (see later section on Portability and Interoperability Issues and Challenges). These various SOAP implementations differ by platform, most notably the Microsoft .NET and J2EE platforms.

One of the primary implementations of the SOAP protocol in the J2EE environment is the Apache SOAP implementation. IBM, an early leader in Web Services development, created SOAP4J, which was subsequently donated to Apache Software Foundation's XML Project. A number of vendors are making use of the Apache SOAP implementation in their systems, thus giving a fairly common level of API usage across multiple different products using the same version Apache SOAP engine. The components of the Apache SOAP implementation include:

- **Apache SOAP 2.0 –** An open source Java implementation of SOAP that includes support for a useful subset of the SOAP 1.1 specification
- **Apache Axis** – A rewrite of the SOAP implementation designed around a streaming (SAX) model to create a more modular, flexible, and better performing SOAP implementation relative to Apache SOAP 2.0.
- **Apache Jakarta Tomcat –** An open source Java Web server and execution environment.
- **Apache Xerces XML Parser –** An open source XML parser

Many systems mentioned in this document utilize the Apache SOAP implementation, while others use a SOAP implementation that has been tested to be interoperable.

Portability and Interoperability Issues and Challenges

There are primarily two issues when dealing with Web Services implementations from different vendors: portability and interoperability. Portability, or porting in its verb form, is the act of moving code between different platforms. In the case of Web Services it simply is impossible to port code from a non-J2EE environment such as .NET to a J2EE environment such as those offered by IBM WebSphere, Sun ONE, and others. However, a more subtle portability issue exists for Web Services vendors on the same platform. Theoretically any J2EE compliant application can run in a J2EE execution environment. However, many Web Services vendors have libraries and SOAP implementations that need to be migrated to the environment in order for them to be run. When evaluating any Web Services implementations, determine whether you can have a disparate execution and development environment and what needs to be "ported" in order for the code to work.

Despite the above, many experts acknowledge that portability will increasingly become less an issue since the promise of Web Services is that any Web Service-compliant component can speak to another regardless of their implementation. This is the very definition of loosely-coupled. As a result, the primary issue is not portability but *interoperability*. Interoperability defines the ability for different implementations to be able to communicate with each other as successfully as they would be able to communicate in a homogeneous environment.

It is in this area that Web Service implementations currently have some challenges that will soon, hopefully, be surmounted. In particular, there has been much criticism that Microsoft .NET SOAP implementations are not 100% interoperable with Apache and other major SOAP implementations. However, one should not lose much sleep worrying over this as a number of major interoperability working groups, task forces, benchmarks, and conformance suites have been created to help move the industry towards global Web Service interoperability. One of these key conformance suites is the SOAPBuilders Interoperability test suite that provides a collection of simple SOAP RPC invocations, in which a client sends a parameter of a certain type (integer, string, etc.) and the server simply returns a parameter of the same type and value.

With these issues in mind, look for greater vendor lock-in for Web Services platforms. With the promise of interoperability, the implementation doesn't matter as much. As a result, there will be less incentive for platform vendors to make their parts interchangeable.

Web Services Registries: UDDI and ebXML

As briefly discussed above, registries and repositories are the main mechanisms for publishing, locating, and finding Web Services in a globally accessible registry. First, why are registries important at all in the context of Web Services?

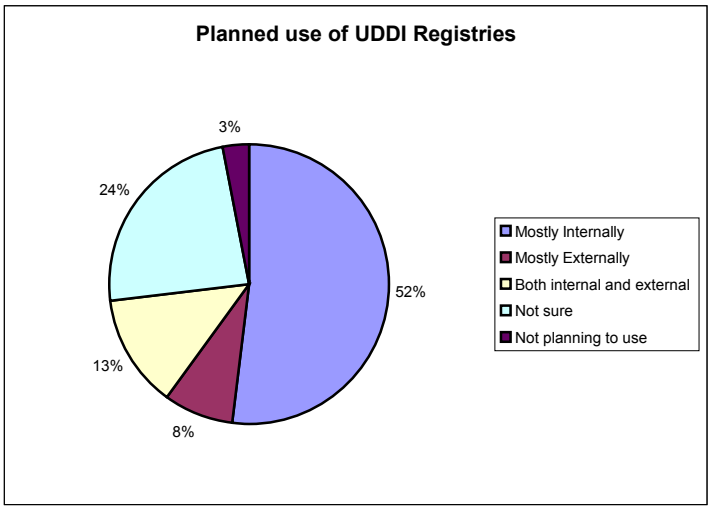
- Since Web Services are loosely-coupled, their interfaces need to be able to be exposed to potential users
- The more Web Services that are present, the greater the need for central registration
- Registries allow for dynamic update of service interfaces without having to notify all potential users of interface changes
- Internally-focused registries allow different business units to discover interfaces for simplifying integration
- Externally-focused registries allow businesses to simplify interaction with partners or provide additional sources of revenue

Sometimes the terms registry and repository are unintentionally interchanged when there is a definite distinction between these definitions:

- The *repository* is the physical data store for profile information. There is no real verb form of the word repository. ("holder of things")
- A *registry* is the mechanism by which repository items and metadata can be registered, located, and queried. Registry is the noun version of the verb "register". ("catalog of things")

UDDI.org describes UDDI mainly from the point of view of exposing Web Services to parties external to a business' organization. The need for registration and location of Web Services is core to the Service-oriented Architecture described earlier. Since the majority of Web Service implementations are inwardly-focused, UDDI registries will mostly be implemented behind the firewall. This leads to the concept of *private UDDI registries*. Private UDDI registries follow the same general architecture of public registries, but the requirements differ in several major ways:

- They are not run by public node operators
- Internal registries have different security and quality of service requirements
- Internal registries do not need to support the same user traffic requirements
- Internal registries are focused on simplifying integration and speeding time-to-market rather than on connecting business partners or providing additional revenue streams
- Internal registries can be built on a wide range of platforms ranging from directory systems to relational databases.



Source: Copyright © 2001, ZapThink, LLC

The ebXML Registry/Repository (RR) mechanism functions in much the same way as is proposed for UDDI, except with a greater focus on business-to-business collaboration. As a result, the ebXML RR specification is more robust and has always had both a public and private registry focus. In ebXML, a company creates a profile for itself known as the CPP, and registers it with the Registry. The CPP contains all the information necessary for potential trading partners to determine which business roles the company is interested in, and which technologies and protocols it can engage in for these various roles. After a company registers itself in the Registry, a prospective trading partner can search the repository for that company's CPP and check to make sure that its profile is compatible with the partner's requirements.

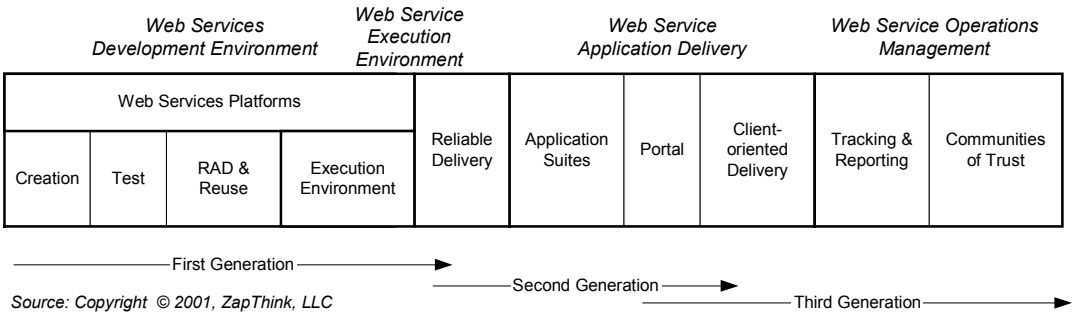
In addition, the ebXML RR specification includes a repository specification that has been specified to be able to store XML as well as non-XML data, while UDDI is primarily an XML-only storage architecture. Despite these differences, UDDI and ebXML are complimentary in their approach and there have been many efforts to synchronize the work efforts going on in these two groups.

Pieces of the Puzzle: Web Services Market Segmentation

Market Overview

Web Services is a market in as much as Client/Server technology is a market – it isn't. It's a computing paradigm and therefore it's not the paradigm as much as the technologies required to implement it that form a market. The market for tools and services for creating Web Services consists of three major segments, each of which is further described below:

Web Services Market Segmentation



Source: Copyright © 2001, ZapThink, LLC

- **Web Services Platform** consisting of a
  - **Web Services Development Environment;** and
  - **Web Services Execution Environment**
- **Web Services Application Development**
  - Business Process Applications
  - Portals
  - Client-side Applications
- **Web Service Operations Management**

Web Services Platforms

The first generation of Web Services implementation is the creation and deployment of the core Web Services components themselves. These are addressed through Web Services Platforms that consist of a development environment, in which Web Services are composed, and a Services execution environment, in which Web Services are run. Web Services platforms allow creation, testing, deployment, discovery, and registration of Web Services.

Web Services Application Delivery

Once Web Services have been created, they need to be aggregated or composed into fully functional applications that serve a specific business need. This represents the "second generation" of Web Services implementation. Web Services Application Development and Delivery suites meet this need by orchestrating, aggregating, and applying workflow and business process to Web Services and presenting the end application to end users through back-end systems, web interfaces, and a variety of display devices.

Web Services Operations Management

The final phase of Web Service implementation is the management of Web Services once they've been deployed in an enterprise. This includes the tracking, reporting, metering, and billing of Web Services in use. This also includes pre-packaged Web Services and communities of trust.

While the above chart shows the overall "pieces of the puzzle" for Web Services, the vendors profiled in this report have a more fine-grained set of distinctions.

Market Segment	Sample Offerings	Description
Web Service Development Environments	Microsoft .NET IBM WebSphere Sun ONE CapeClear HP Web Services Platform IONA Orbix E2A The Mind Electric GLUE Systinet WASP	Allow creation of Web Services from scratch or wrapping of existing components. Some have an IDE and Application Server execution environment and some don't.
Web Service RAD & Reuse Environments	AltoWeb Infravio LogicLibrary Shinka Technologies VelociGen	Provide an environment for rapid development of Web Services and facilitate component reuse
Web Service Execution Environment	BEA WebLogic Oracle 9i App. Server	Provide an execution environment for Web Services. Many of the Development Platforms also have an execution environment.
Reliable Delivery	Flamenco Networks Grand Central Kenamea Slam Dunk Networks Vordel	Provide security, reliability, and transaction control for Web Services
Web Service Transport Protocols	BEEP IBM HTTPR Jabber	Provide alternatives to HTTP and SMTP for Web Service delivery
Web Service Application Suites	Attunity Avinon Bowstreet Instantis	Assemble and orchestrate Web Services into applications that can be deployed on the back-end or with user interface
Web Service-enabled Portals	DataChannel Epicentric Plumtree	Provide a portal environment for the deployment of Web Service applications
Client-oriented Web Services Delivery	Altio Curl	Provide presentation layer and client-side (thin or thick) applications for interacting with Web Services
Web Service Operations Management	MetraTech Softricity	Value-added operations for management of Web Services including Reporting, Billing, Metering, and QoS.
Web Service Communities of Trust	XMethods	Communities for sharing Web Services.

Web Services Platform: Development and Execution Environment

In order to make use of Web Services, the first necessary steps are to create Web Services, either from scratch or by wrapping existing functionality (such as EJB or COM components or other legacy functionality).

A Web Services "Platform" actually consists of two major elements: a development environment and an execution environment. The features of each are described below:

- Development Environment
  - Creation of Web Services
    - Integrated Development Environment (IDE)
    - Support Wrapping of existing application logic as well as de-novo creation
    - Graphical component assembly
    - Abstraction from implementation specifics
  - Testing of Web Services
  - Deployment of Web Services to Runtime Destination
  - Discovery, Registration, and Reuse of Web Services
- Execution Environment
  - Service execution environment
    - Application Server environment
    - Web Service implementations
    - Connection to Legacy systems and data
  - Secure, reliable transport

The Development Environment

Part of the reason for the rapid growth of Web Services is due to the fact that many existing technologies can be simply extended to provide the XML-based interface required for Web Service communication. As a result, many of the features and requirements of traditional application component development can, and have, been applied to the development environment. This includes rich Integrated Development Environments (IDEs), support for wrapping existing application functionality and logic, visual component assembly, testing, and the discovery, registration, and reuse of existing components.

The Execution Environment

Once a Web Service has been created, it makes sense that it needs to be executed in a runtime environment. Just as development tools for "traditional" components have been applied to Web Services, so too have runtime and execution environments. In particular, the popular Application Server that has been used to provide an environment for Web applications has been applied to Web Services. As a result, many of the players in the application server space have become front-runners for delivery of Web Services as well.

Linking between Execution and Development Environments

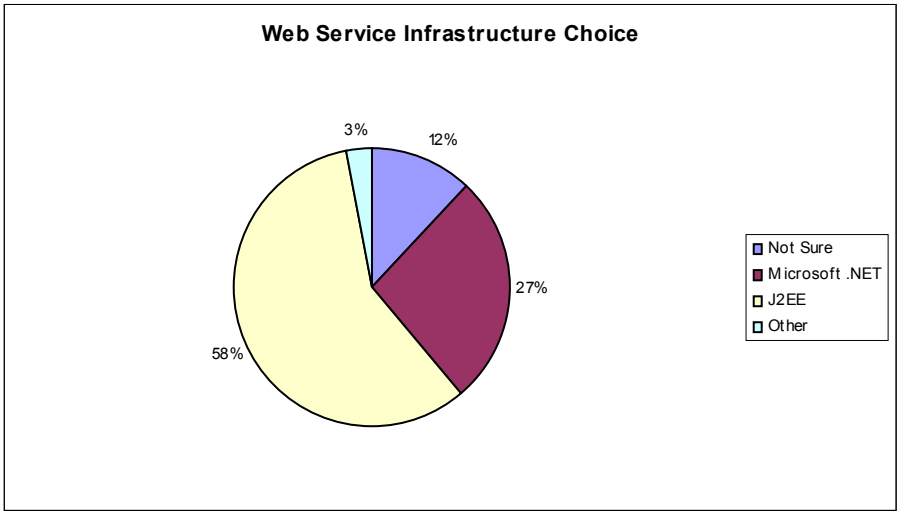
While theory should allow us to create development environments that can produce Web Services that can be run in any execution environment that can consume the appropriate class objects, the practical realities are that development and execution environments are inherently linked together. This means that a purchase of a development environment will necessitate either purchase of execution environment for processing those developed Web Service components or the installation of runtime libraries on the appropriate application server platform.

One of the main reasons for this is that there aren't any standard interfaces between development tools and SOAP engines. There are a number of SOAP implementations, each with

their own APIs. Although SOAP messages have been defined in a standardized manner, the APIs for accessing these messages have not. In addition, tools that "wrap" existing COM or EJB components contain their own runtime libraries that necessarily need to reside at the server location at time of invocation. While J2EE-platform based utilities may have a degree of flexibility in their deployment, there is by no means any interest or capability in having .NET build services be able to be executed on J2EE execution environments. A potential solution to this "portability" issue, on the J2EE platform at least, is the development of a standard API for accessing SOAP engines through the JAX/RPC , JAXM, and JAXR efforts. Once standard API has been defined, then the promise of being able to create a Web Service in any J2EE development environment and implement on any execution environment can be realized, unless client libraries are required. So, while development and execution environments form logically separate markets, the reality is that the Web Services Platform is a necessary combination of both aspects.

Infrastructures of Choice: Microsoft .NET or Java2 Enterprise Edition (J2EE)

A developer who chooses to implement Web Services has a variety of solutions at his or her fingertips. As long as standards are adhered to, Web Service applications can be built in Visual Basic, Java, Fortran, Cobol, Perl, or even assembly language. However, when it comes to robust commercial implementations supported by wide vendor communities (especially application server vendors), there are typically two major platforms that are usually considered: the Microsoft .NET Framework and the Java 2 Enterprise Edition (J2EE) architecture.



Source: Copyright © 2001, ZapThink, LLC

Microsoft .NET Platform

Microsoft has entered the Web Services spaces with not a whimper, but a bang. Their .NET platform is a comprehensive strategy to deliver Web Services as part of the core architecture of their product lines going forward. In general, their strategy encapsulates the following major pieces:

- An IDE for development of Web Services in the form of Visual Studio.NET and associated development tools and libraries
- An execution environment including major enhancements to the Windows 2000 line (now known as Windows. NET Server)
- Enterprise servers that are Web Services enabled (Exchange, SQL Server, etc.)
- Pre-packages Web Services and enabling components (My .NET Services, Passport)

While Microsoft is definitely casting most its chips in favor of this new Web Services-oriented strategy, its approach has it traveling in a separate direction than its competitors. While surely .NET services will be interoperable with non-.NET created Web Services, they will only be able to be run on the .NET platform. This more general portability issue may cause concern with many developers who will surely be looking to leverage their skills more widely.

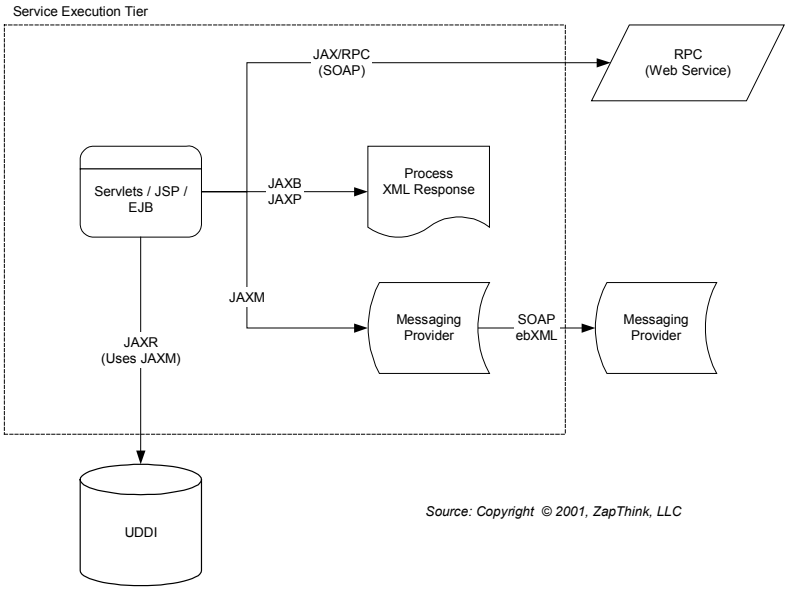
J2EE Platform

The Java2 Enterprise Edition (J2EE) platform has emerged as a major force shaping the development of robust web-delivered applications. Emerging from its early incarnation as a "cute" applet delivery tool, Java has become a major part of most enterprise-class application delivery strategies. As a result, J2EE has emerged as a major contender for the development and delivery of Web Services applications.

It goes without saying that the majority of Web Services vendors that are not on the .NET platform are on the J2EE environment. Vendors are featured in the vendor profile section.

On the J2EE platform, a number of choices can be made for development and delivery of Web Services. Below is a diagram showing the basic Java APIs and their interaction with different elements of Web Services delivery.

Java Web Service Technology Interaction



Web Service Transports

The concept of messaging is simple: get a message from point A to B. However, there are many notions of transport, all of which are relevant to Web Services. As a result there are three major types of message transport:

- **"Wire-level" transport** – the protocol required to establish a connection and provide a passage or tunnel for XML-formatted data to pass. (examples: HTTP, SMTP, FTP)
- **Message-level transport** – the messaging envelope that surrounds relevant XML data and provides document-level routing, security, transaction, and privacy controls. (Examples: SOAP, ebXML TRP layer)
- **Reliable Messaging** – providing a guaranteed mechanism for messages (formatted as SOAP or otherwise) to reliably reach their end destination fulfilling the business requirements of the message-level transport. (Example: the EDI VAN)

Since we are talking specifically about Web Services, we divide this section into a discussion of Wire-Level Transports and Reliable Messaging. The message-level transport for Web Services is SOAP.

Wire-Level Transport

There are a number of existing protocols that are being used to transfer XML-based messages from point to point. They include:

- **HTTP** — The Hypertext Transfer Protocol that forms the basis for most, if not all, web server communication. Includes its secure-sockets variant HTTPS.
- **SMTP** — The Simple Mail Transfer Protocol that forms the basis for most email exchange on the Internet.
- **FTP** — The File Transfer Protocol that serves as one of the main methods for exchanging binary and ASCII files on the Internet.
- **Message Queues** — Vendor proprietary technology (such as IBM's MQ Series and Microsoft's MSMQ) that are responsible for reliably shuttling messages for applications and message-oriented middleware (MOM).

While these protocols form the bulk of most Internet and behind-the-firewall messaging, they are either not robust or reliable enough for wide scale Web Services requirements or are proprietary in nature. As such, a new breed of Internet protocols that aim to assist in the reliability and flexibility of Web Services delivery are being proposed.

IBM's Reliable HTTP Protocol (HTTPR) and Dependency Spheres

In order to solve the needs of reliable Web Services delivery, IBM is proposing reliable extensions to the ubiquitous hypertext transport protocol (HTTP). The reason for extending HTTP rather than creation of a new protocol is that HTTP is widely deployed and understood by developers as well as network administrators. This allows companies to make use of reliable messaging while simultaneously retaining their investments in existing web technologies as well as leveraging HTTP's capabilities for security, sessions, proxies, and firewall support.

HTTPR works by ensuring that the client and server retain state and can ensure that a given transaction has reliably occurred or not. This requires the capabilities of both ends to store their message for a given amount of time. HTTPR has not been widely implemented and it is not clear if their own products plan to support the protocol in the near future.

IBM has also introduced a solution for implementing distributed transactions through a notion called "dependency spheres". Dependency spheres are a new type of transaction context that

allows both synchronous and asynchronous distributed messaging style exchanges to occur within a single transaction. The implementation of this is through a third-party Web service that provides a transactional context that ensures that transactions occur even in distributed heterogeneous environments. Dependency spheres are still a research project and so aren't available even for alpha testing.

Jabber

Originally developed to enable Instant Messaging across different messaging platforms, the **Jabber** Peer-to-Peer (P2P) messaging system is being applied to the transmission of Web Services instead of "chat". The system is actually based on a client/server model that allows for real-time messaging (not store-and-forward) between clients that can then alternatively have a direct peer-to-peer conversation as required. All communications are coordinated through a server that is aware of when clients are connected to the network through a notion called *presence*. This knowledge of availability helps to deliver messages to clients and allows transmission of application-specific messages.

Users on the system are associated with particular servers and each user is assigned an ID that is similar to email addresses. Clients can be designed very simply since all that is needed to communicate with a Jabber server is TCP and XML. Jabber is an open-source based effort and there are a number of efforts under way to use Web Services as a message type being communicated with clients. Currently there are projects under way for SOAP and XML-RPC messaging over Jabber, but there have yet to be any real implementations of these projects. The future of Jabber really lies in its adoption by other parties and the willingness of developers to use the Jabber protocol instead of HTTP and other delivery mechanisms.

Blocks Extensible Exchange Protocol (BEEP)

The Blocks Extensible Exchange Protocol (BEEP, formerly BXXP) is intended to be a framework upon more specific protocols can be developed, such as those relevant for Web Services exchange. Their foundation is that good protocols are hard to design, where good means well defined, complete, parsable, extendible, and easily written to. A specific BEEP instance takes the form of a "profile" that outlines a specific channel for communication and management of that channel. BEEP protocols can be message/reply, message/error, and message/multiple reply in format.

BEEP is forming the basis for other protocols such as the Internet Messaging Exchange Protocol (IMXP), which is similar in some aspects to Jabber. However, BEEP has yet to find any specific widespread adoption within the Web Service community and there haven't been many Web Service-specific protocols in development. As such, it remains a great concept but only a proposed framework rather than something that can be adopted in any short-term time frame.

Reliable and Secure Messaging

The idea behind reliable and secure messaging is to insure that the following criteria are met in sending a message between parties:

- The message was actually sent
- The message was actually received
- Only one copy was received by the intended recipient
- The message follows a given transactional order
- The security or privacy of the message has not been compromised in the process

And, optionally, some features of "conditional" messaging which include:

- The validity of messages for a given time frame
- Content and context-aware messaging

- Messages intended for certain recipients only

Why isn't HTTP or the other Wire-Level protocols good enough? Because their main goal is to facilitate the process of connecting parties together to communicate, but not establish any ground rules for that communication. While HTTPR goes to some extent to make sure that HTTP servers are reliable, there is no guarantee that the HTTP server itself will be up and running. What is needed is a more robust, reliable, secure, and transactional protocol to insure that *business processes* occur and not just messaging.

The other reason for not adopting the new protocols is that most, if not all, of the above protocols have yet to really be implemented. They are not quite ready for prime time, and as a result, those who have needs *today* for reliable messaging delivery should seek a solution from one of the vendors below.

There are two major ways to implement Reliable and Secure Messaging:

- Robust messaging technology implemented by individual communicating parties (PKI, Entrust, Vordel)
- Managed Reliable Delivery Networks (RDN) that provide a controlled, secure environment for message exchange. (Slam Dunk Networks, Flamenco, Grand Central, Kenamea)

Criterion for Evaluation of Reliable Delivery Networks

Since RDNs are not open standards but rather business offerings, not all solutions will be the same. There will be many differences in feature sets, technical capabilities, and levels of service. Therefore it is important to evaluate the network with criteria in mind for how it will be utilized and implemented.

Features of RDNs should include:

- Support for Synchronous and Asynchronous messaging
- Routing of messages
- Encryption, Access control and authorization
- Reliability and non-repudiation
- Transaction control
- Polling of available messages by recipient
- Registry and Discovery of clients

Business-level features of RDNs should include:

- Same criterion you would use to evaluate an ISP or hosted offering
    - Global presence and availability
    - "Five 9's" of reliability: 99.999% uptime
    - Distributed and robust network infrastructure
    - Multiple ISP pipes
    - 24/7 Customer Support (saying 365 is redundant) and NOC management
- Support for Trading Network
    - Many different message types and host systems; especially Web Service standards
    - Can handle peak message loads (byte size and volume)
    - Support for management of trading partner networks
- Guaranteed Delivery
    - Configurable and predictable transmission timeframes
    - Real-time and store-and-forward messaging
    - Alerts and notifications on delivery or failure
- Secure
    - Levels of security configurable for each trading partner

    - Managed certificates
    - Various levels of security at message, transport, and user level
- Audit Control and Backup

In many ways, the RDN is similar to the EDI VAN and has to meet many of the requirements and features that the system offered to EDI users. What makes an RDN compelling in a Web Services scenario is that by utilizing a trusted intermediary, the Web Service does not have to worry about the security, reliability, and robustness aspects of delivery, thus keeping the Web Service lightweight and simple in design.

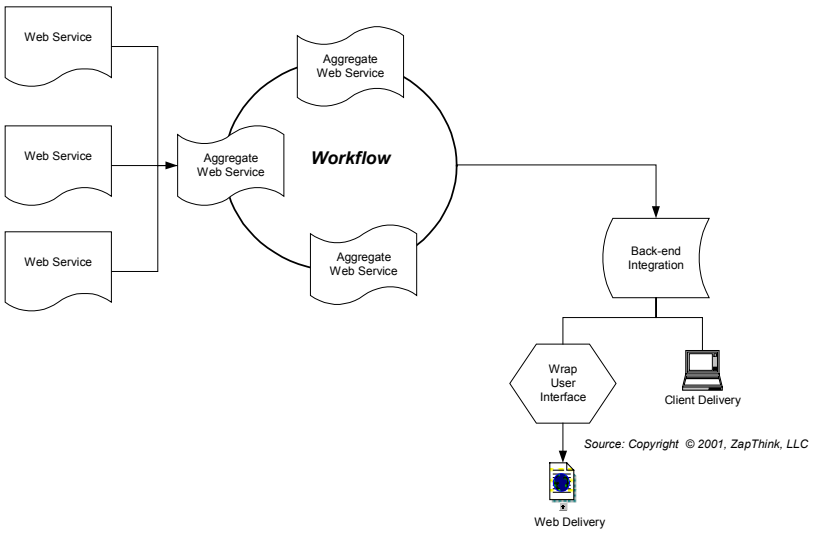
Web Service Application Development and Delivery (WSAD)

Once the Web Service has been created and a method has been determined for shuttling it between end-points, it becomes necessary to aggregate or conglomerate multiple Web Services into a usable application. This is known as Web Service Application Development and Delivery (WSAD). Aspects of WSAD include:

- Choreography and orchestration of Web Services into useful applications and business processes
- Inclusion of Workflow and Back-end Business Process
- Output of Web Services
- Integration of Web Services with back-end processes; or
- Formatting of Web Services for display
    - Web / Portal interface
    - Client / Device interface

In the process of building an application, one needs to combine web services together in order to form a new, aggregate web service or to utilize the web service as a portion of back-end process or interface logic. Workflow is defined as the organization of processes into a well-defined flow of operations to fulfill a business need. The distinction between output mechanisms is clear: in back-end processes, we are concerned with machine-to-machine interaction, whereas with user interface processes, we are concerned with human-to-machine interaction. The below diagram illustrates these concepts. However, since the output type is linked to the mechanism for aggregating Web Services, the vendors and products featured below are grouped by final output destination: back-end, web-based, and client-side.

Web Service Application Development & Delivery



Web Services Application Delivery Suites (WS ADS)

One class of WSAD are the application delivery suites that orchestrate Web Services to produce applications that are either used in back-end systems or to drive user interaction. These application suites are equally suited to machine-to-machine interaction as well as human-to-machine interaction.

Key features of Web Services ADS include:

- Orchestration of Web Services into aggregate or composite web services
- Application of business process, workflow, and rules to orchestration
- Integration of Web Services with back-end and legacy systems
- Use of Web Services to drive back-end transactions or user interactions across multiple device types

Web Services-enabled Portals

Another class of WSAD, and possibly a subclass of ADS, is Web Service-enabled Portals that serve to expose aggregated Web Services to users through a modular, personalized, and flexible user interface. Portals have long been used to deliver personalized content to Internet, Extranet, and Intranet users. These same systems are now being used to deliver aggregate Web Services to these same users. Historically, the main interface for portals has been the web, but increasingly portals are being applied to deliver content to alternate display devices such as cell phones and PDAs.

Key features of Portals include:

- Aggregation and orchestration of Web Services into flexible, personalized delivery to users
- Integration with content management systems
- Integration of Web Services with back-end and legacy systems
- Delivery of Web Services to multiple device and display types

Presentation Layer Web Services Delivery

While one half of the term "Web Services" refers to the web, there is no reason why Web Services need to be deployed solely over web browsers or over the HTTP protocol. In fact, there

are a number of offerings that aim to reach much better efficiency gains by using a richer client rather than using the web and provide solutions at the presentation layer that are distinct from portal solutions.

What's wrong with the web, even with Java, ActiveX, and richer HTML?

- Every request for a resource is a round-trip requiring bandwidth and processing resources
- Client systems have richer user interface possibilities than purely web-browser based
- Distributing the user interface lessens load on web servers.
- Data updates are received live by clients and instantly updated.
- Java and ActiveX solutions can be too large, bulky
- HTML, JavaScript, Java version dependencies
- Support for offline applications

Features of Presentation Layer delivery include:

- Use of non-Web technologies and lightweight XML transfer for Web Services presentation and delivery
- Provide an environment for creation of user interfaces
- Provides rich user interaction on par with current client/server applications

Web Services Operations Management

The next generation of Web Services products and services are involved with the management of existing Web Services. These Web Services Operations Management (WSOM) technologies are concerned with value-adding Web Services so that they can have the same high quality and performance characteristics as existing computing technologies.

Tracking and Reporting

One key aspect of WSOM is the tracking and reporting of Web Service components for the purposes of identifying Web Service resources in use and for billing and metering purposes. Features of tracking and reporting services include:

- Identification of Web Services resources in UDDI registries or on the network
- Tracking usage of Web Services
- Billing and metering of Web Services

Runtime Management and Quality-of-Service

Of major concern to organizations that have or will have a vested interest in Web Services implementations in mission-critical implementations, such as transactional systems in financial applications, is the ability to manage the quality and availability of the Web Service to consumers. It is vitally important to make sure that Web Services can respond to requests and perform their operations. As a result, WSOM tools that focus on runtime management and QoS have features that include:

- Managing uptime for services
- Load balancing, QoS, and failure prevention
- Version control

Web Services Communities

With the development of Web Services comes an emerging after-market in the exchange and sale of pre-packaged Web Services that can either be accessed via the Internet or hosted locally. These Web Services provide a host of application functionality that has long been the promise of externally-focused Web Services implementations. These Web Services can be accessed for free or made available for a fee. These trends are covered in the Trends and Directions section below.

To date, there are a limited number of firms offering Web Services globally to interested parties. Two of the more notable firms include:

XMethods – A community web site that lists publicly available Web Services. Currently, many of the registered services are trivial in nature, but it is hoped more robust business-based services will be offered soon.

Microsoft .NET My Services – Formerly named "Hailstorm", the .NET My Services initiative by Microsoft is releasing a number of pre-packaged Web Services aimed at usability and personalized information that range in functionality from profile and contact information to calendars and wallet information. These services are being made available for a fee that has yet to be fully resolved. Both users and developers will pay to access these services.

Trends and Directions

Competitive Pressures

The number of vendors entering the Web Services market is increasing at an astounding pace. This is a testament both to the business opportunity that Web Services presents as well as the ability to easily integrate and make use of the technology. However, the emergence of these vendors causes some competitive pressure that may result in a "boom/bust" cycle. New competitor entrants include:

- Platform vendors
- Middleware vendors
- Emergent Web Services startups
- Application Server vendors
- EAI and data integration vendors

Federated Identity (Single sign-on)

The idea behind federated identity is that systems can abstract user information into secure, private identity containers that can then be exchanged between systems avoiding the necessity for maintaining multiple logins and profiles for different systems. This allows users to easily move between systems without running into security and authorization difficulties and differences. Federated identity also provides a simple and centralized manner for administrators to manage user identity and privileges across multiple systems and enterprises. There are two major Federated Identity efforts:

Microsoft Passport – Offered by Microsoft and a basis for My .NET Services, Passport provides a means for single sign-on and federated identity within the .NET framework.

The Liberty Alliance Project – Headed by Sun, the Liberty Alliance Project provides federated identity and single sign-on in the J2EE environment. Includes decentralized authentication and open authentication for any type of device or system.

Fee-based Web Services

With the development of remotely executable and deployable services comes the desire to generate revenue from the sharing of these modular functionality components. The current focus of most Web Services implementations is on cost savings and efficiency improvements, but soon attention will shift to revenue-enhancing opportunities. As a result the concept of fee-based Web Services has gained interest. Fee-based Web Services allow organizations to make functionality available to users for a fee. Besides this concept, not much else has been standardized as far as commercial offering of Web Services. In particular, there is not yet an accepted pricing structure. Some concepts around Web Services pricing models include:

- **Fee-for-usage** – Per transaction or invocation charges.
- **Subscription** – Charges based on a period of time, such as yearly, monthly, or even daily.
- **Shrink-wrapped** – Services offered for a one-time charge that can be invoked remotely or installed locally.

The offering of commercial Web Services has resulted in the notion of "core" or "utility" services as well as "application" services. Utility services serve as the plumbing for Web Services and include security, transformation, logging and reporting, authorization control, user management, account management., and billing and metering. Application services run the gamut of usage

scenarios such as weather, stock market, communication, and other offerings. However, with the sale of Web Services as a commercial offering come a number of challenges

- Service Level Agreements – with the sale of Web Services needs to come assurances that the service will meet agreeable usage quality.
- Security – Solutions will need to be in place that adequately address security concerns.
- Billing and metering – There need to be appropriate technologies for billing and metering
- Performance – Performance will need to be guaranteed
- Transactions – Some transactions need to exist over days or months.
- Jurisdiction and point of control – What sort of legal, privacy, regulatory, and other issues will come to the surface when Web Services are implemented in a distributed fashion?

Other challenges include:

- Some existing business processes do not easily migrate to Web Services.
- Businesses need to come up with new pricing models.
- Web application server in and of themselves are not sufficient to deploy, host, monitor, and bill for Web Services
- An application's dependency on Web Services can have dramatic performance challenges.
- A critical mass of reusable Web services must first be reached before widespread distributed computing can truly occur.

Challenges

Web Services as an emerging market and a relatively new technology has a number of challenges that need to be addressed for long-term market viability. These challenges stem from:

- Reliability and Security issues
- Standards development and interoperability
- Scalability and Performance

Reliability & Security

While addressed by a number of the major technology vendors and protocols discussed in this report, there have yet to be a set of well-accepted standards or vendor scenarios that address reliability and security. In order to make a robust system, a hodge-podge of standards, vendor offerings, and server-side tools need to be implemented. There is no doubt that this will be improved upon in the months or years ahead, but until then, Web Services will be a "Wild West" frontier land fraught with challenge and peril.

Standards & Interoperability

The standards themselves pose challenges to those who are implementing Web Services *today.* As discussed earlier, there are a number of interoperability and portability issues with regards to Web Service implementations. However, it is expected that many of the interoperability issues will be resolved over the next 12 months.

However, SOAP is not the only RPC-like XML protocol. Others efforts around XML-based RPC mechanisms include XML-RPC, WDDX, Wf-XML, and XMI. There is question as to how well these specifications will succeed and thus need to interact with the SOAP-based Web Services stack. More importantly, ebXML has become a major force for supplying an end-to-end protocol for B2B exchange. Essentially, ebXML specifies almost every aspect of the e-Business transactions, ranging from service definition to describing entire business process workflows. In addition, ebXML covers a broad range of aspects relating to reliable messaging, security, transactions, message packaging, description of services, description of service capabilities, sequencing of

messages, message workflow and orchestration, agreements between various trading partners, and other issues.

In many ways this sounds exactly like the Web Services story. The end result is that there are a number of conflicts and collisions between the groups working on ebXML and Web Services standards. There has been some attempt to synchronize the efforts, but a number of major conflicts still exist. What is important to realize, however, is that ebXML and Web Services are approaching the same goal from very different points of view. The ebXML specification works from the top down, identifying the full range of requirements necessary for successfully conducting electronic-business over the Internet and then working to implement specifications that meet those requirements. The Web services architecture starts bottom and works its way up, creating specifications to meet individual messaging and functionality requirements and then building upon that foundation. As the two work towards each other, the differences in implementations will need to be worked out.

Scalability and Performance

While Web Services leverage much existing technology, their performance in mission-critical and high-performance systems have yet to be fully proven. May Web Service platform vendors are indeed attempting to make sure that their end products are scalable in these environments and there is no doubt that we will see high performance Web Services implementations. However, the current number of case studies for these high-performance implementations are limited.

It is also a difficult proposition to test Web Services across multiple execution environments and SOAP implementations. Therefore, a number of challenges exist not only in scaling the Web Services themselves, but in testing their interaction with other services.

Vendor Profiles

Web Services Development Platforms

IBM
Please see ZapNote ZTZN-1050

Microsoft
Please see ZapNote ZTZN-1066

Sun Open Net Environment (ONE)
Please see ZapNote ZTZN-1093

Cape Clear
Please see ZapNote ZTZN-0120

Hewlett-Packard (HP) Web Services Platform
HP has discontinued its Web Services development products. Please see ZapNote ZTZN-1048.

IONA Orbix E2A e-Business Platform
Please see ZapNote ZTZN-0140

The Mind Electric GLUE
Please see ZapNote ZTZN-1098

Systinet (formerly Idoox) Web Applications and Services Platform (WASP)
Please see ZapNote ZTZN-1096

Web Services Execution Environments

BEA WebLogic
Please see ZapNote ZTZN-1009

AltoWeb Application Platform
AltoWeb is no longer in business

Infravio
Please see ZapNote ZTZN-0226

Shinka Technologies
Please see ZapNote ZTZN-1090

VelociGen
Please see ZapNote ZTZN-1101

Reliable Delivery Networks (RDN)

Flamenco Networks
Please see ZapNote ZTZN-0150

Grand Central Web Service Network
Please see ZapNote ZTZN-0623

Kenamea
Please see ZapNote ZTZN-1058

Slam Dunk Networks
Slam Dunk Networks is no longer in business

Vordel TalkXML
Please see ZapNote ZTZN-0238

Web Services Application Suites

Attunity eBusiness Integration Suite
Please see ZapNote ZTZN-0138

Avinon NetScenario
Please see ZapNote ZTZN-0108

Bowstreet Business Web Factory
Please see ZapNote ZTZN-1015

Instantis SiteWand
Please see ZapNote ZTZN-1051

Web Services-enabled Portals

Epicentric
Please see ZapNote ZTZN-0104

Plumtree
Please see ZapNote ZTZN-1079

DataChannel
DataChannel is no longer in business

Presentation Layer Web Services Delivery

Altio
Please see ZapNote ZTZN-1005

Curl

Please see ZapNote ZTZN-1025

Web Services Operations Management

LogicLibrary

Please see ZapNote ZTZN-0181

Methodology

ZapThink collected the information above in most cases by conducting phone briefings directly with individuals at the vendors profiled in this report. For those vendors that were unable to speak with analysts, ZapThink used publicly available sources such as web sites, marketing literature, financial filings, and reviews conducted by other journalistic or analyst agencies.

Market statistics, numbers, and positioning diagrams were generated by examining public information gleaned from financial filings and web sites or through direct interviews with profiled vendors. For vendors that didn't provide ZapThink with specifics as to private business information, analysts compared the firm against others in its class that shared similar characteristics and assumed similar business figures. In those cases, ZapThink may be off a significant percentage in a given market size. Care was taken to lower the weighting that these "estimated" figures impacted on overall aggregate statistics.

About ZapThink, LLC

ZapThink is an IT market intelligence firm that provides trusted advice and critical insight into XML, Web Services, and Service Orientation. We provide our target audience of IT vendors, service providers and end-users a clear roadmap for standards-based, loosely coupled distributed computing – a vision of IT meeting the needs of the agile business.

ZapThink's role is to help companies understand these IT products and services in the context of SOAs and the vision of Service Orientation. ZapThink provides market intelligence to IT vendors who offer XML and Web Services-based products to help them understand their competitive landscape and how to communicate their value proposition to their customers within the context of Service Orientation, and lay out their product roadmaps for the coming wave of Service Orientation. ZapThink also provides implementation intelligence to IT users who are seeking guidance and clarity into how to assemble the available products and services into a coherent roadmap to Service Orientation. Finally, ZapThink provides demand intelligence to IT vendors and service providers who must understand the needs of IT users as they follow the roadmap to Service Orientation.

ZapThink's senior analysts are widely regarded as the "go to analysts" for XML, Web Services, and SOAs by vendors, end-users, and the press. They are in great demand as speakers, and have presented at conferences and industry events around the world. They are among the most quoted industry analysts in the IT industry.

ZapThink was founded in October 2000 and is headquartered in Waltham, Massachusetts. Its customers include Global 1000 firms, public sector organizations around the world, and many emerging businesses. ZapThink Analysts have years of experience in IT as well as research and analysis. Its analysts have previously been with such firms as IDC and ChannelWave, and have sat on the working group committees for standards bodies such as RosettaNet, UDDI, CPExchange, ebXML, EIDX, and CompTIA.

Call, email, or visit the ZapThink Web site to learn more about how ZapThink can help you to better understand how XML and Web Services impact your business or organization.

ZAPTHINK CONTACT:
ZapThink, LLC
11 Willow Street, Suite 200
Waltham, MA 02453
Phone: +1 (781) 207 0203
Fax: +1 (786) 524 3186
info@zapthink.com

ZapThink, LLC • 11 Willow Street, Suite 200 • Waltham, MA 02453 • info@zapthink.com • www.zapthink.com