Longevity of Web-Services – Will it justify the expectations?

Abstract

There is a claim that Web Services technologies will not enable anything new, as they do not offer anything over previous technologies making similar claims (i.e., DCOM, CORBA, and XML-RPC). I take a stance against that and argue that Web Services technologies have certain distinguishing characteristics about it that will enable it to grow from strength to strength. I provide a brief history of the development of Web Services and also compare the other technologies in this domain (CORBA, COM/DCOM, XML-RPC) with Web Services and highlight the differences. I conclude by extending my argument to claim that the best of web services is yet to come and that it has the potential of being a 'disruptive technology'.

Introduction

What are 'Web Services'? There seem to be different opinions on what exactly constitutes Web Services.

"*Loosely coupled software components that encapsulate discrete functionality and that are accessible over standard Internet protocols.*" - The Stencil Group
"*Web Services are a new breed of web applications. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the web.*" – IBM Web Services Tutorial

While the author concurs to a great level with IBM's definition on Web Services, what follows below in the author's humble opinion, clarifies further the definition of Web Services closest to what it started off being:
The term Web services describes a standardized way of integrating Web-based applications using the Extended Markup Language (XML), Simple Object Access Protocol (SOAP), Web Service Definition Language (WSDL), and Universal Description, Discovery, and Integration (UDDI) open standards over an Internet protocol backbone. XML is used to tag the data, SOAP is used to transfer the data, WSDL is used for describing the services available and UDDI acts as the standard registry service and is used for listing what services are available. Under web services, a program could be viewed as the building block for a more complex program and this continues in a recursive fashion. It indicates a distributed technology, which entails bridging across different technologies, platforms, and enterprises. They are neutral with respect to object models and types, and provide asynchronous communication of high-level, semantically rich, XML business documents.

Web Services has today diversified into various applications like e-business applications (for instance eBay API, Amazon API), searching tools (Google API), and even programming assignment cheat checkers (MOSS – Measure of Software Similarity, developed by UC, Berkeley). One can search http://www.uddi.org or http://www.xmethods.com for web services currently implemented by various individuals/companies, sorted categorically. Other simple web services that one could easily find on the web include web services to return the current time and web services to validate US Bank routing numbers.

Some of the salient features of Web Services are:

- Software as a service - As opposed to packaged products, web services can be delivered and paid for as streams of services and allow ubiquitous access from any platform. Web

Services allow for encapsulation. Components can be isolated such that only the business-level services are exposed. This results in decoupling between components and more stable and flexible systems.

- Dynamic Business Interoperability **-** New business partnerships can be constructed dynamically and automatically since Web Services ensure complete interoperability between systems
- Accessibility **-** Business services can be completely decentralized and distributed over the Internet and accessed by a wide variety of communications devices
- Efficiency **-** 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 tracking and fixing. The overall result is risk reduction and more efficient deployability.

Early Initiatives

The ability to perform binding at run time was attained in the mid-80s and early 90s with 'componentized' objects, namely the System Object Model (SOM) from IBM, the Component Object Model (COM) from Microsoft and Common Object Request Broker Architecture (CORBA) in the Unix universe. This binding could only occur within a single software 'universe', e.g., a computing system had to be all Microsoft using COM, or all Unix using CORBA. 'Cross-universe' systems could be built, fairly expensively, using proprietary connectors, each job being one-of-a-kind.

All these efforts had some shortcomings that prevented them from revolutionizing the industry as such. At the same time, e-mail and the Web were proving themselves the most successful distributed architectures ever, and designers sought distributed development technologies that provided the looser coupling of messaging technologies and the ubiquity of the Internet. Standards that were embraced by all major vendors were another wish-list item as they would reduce the risk associated with choosing such technologies. Also during this period other more specialized distributed technologies were emerging.

So, as the new millennium approached, the stage was set for a new generation of distributed computing, based on information-systems needs and experience with network technologies to date. The prevalent needs were:

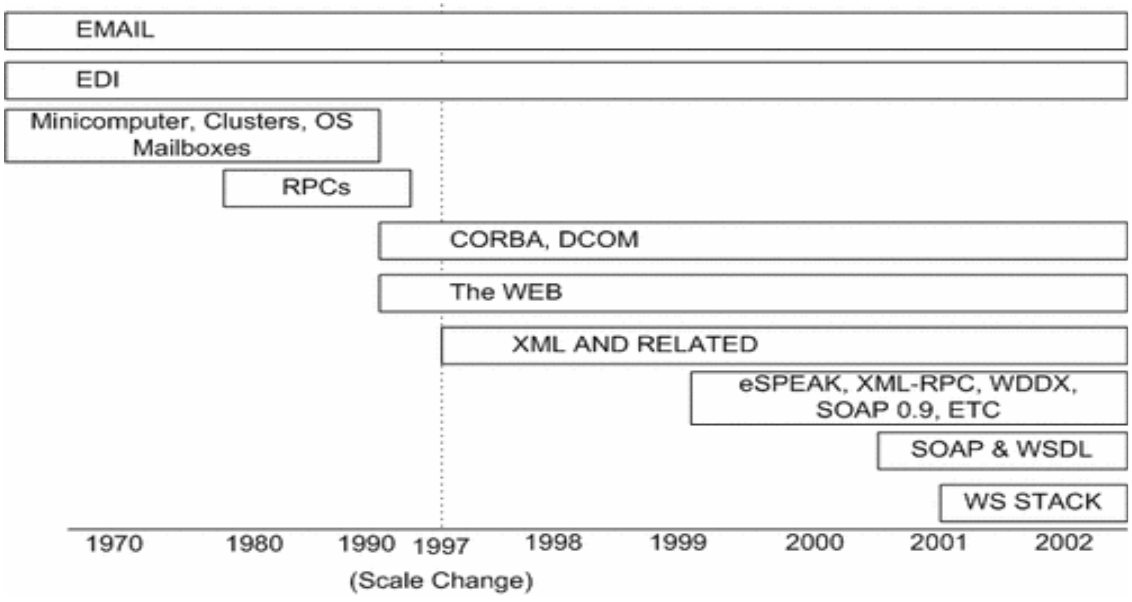
- Suitability both for distributed operation within an application, and the use of generic services across applications. In other words: the ability to support both software developers and systems integrators.
- Suitability for exchanges within an organization and between organizations, requiring cross-platform support and a data-driven focus.
- Concordance with existing Internet infrastructure as much as possible. Ability to scale as the number of nodes, heterogeneity of nodes and the complexity of each node's needs increase.
- Strong support for internationalization.

- Tolerance of failure. Networks where nodes are very tightly coupled together often suffer catastrophic failure when one node goes down. This is a serious problem for heterogeneous networks.
- Strong support in general software development and business workflow management tools from a rich choice of vendors.
- Suitability for the most trivial request/response exchanges as well as handling the most sophisticated orchestration, transaction and security concerns where necessary.

In the early to mid 90s, HP Labs began to explore how to reduce the well-known technical and cost problems of distributed systems. The resulting design principles which led to HP's e-Speak, released in late 1999, addressed most of the needs outlined at the end of the last section, and e-Speak emerged as probably the first Web Services technology, and certainly the first commercially marketed Web services technology. It used generic protocols such as HTTP and an XML data representation to treat all manner of networked systems as "e-services" into which one could rapidly plug data streams. Unfortunately, because its vision is far more coherent than the current state of Web Services, HP suppressed e-Speak in favor of a more mainstream Web Services offering.

The same insight into how to use HTTP and XML technologies to meet the above needs came to the less corporate UserLand community, whose leader, Dave Winer, led the development of XML-RPC, a very simple (the specification is only a few pages) system for calling functions on a remote server. XML-RPC is still very popular, especially in the open-source community where peer pressure and the development of many open and license-free implementations has led to high levels of interoperability, and low cost. Perhaps inevitably, given its shoe-string roots and early arrival, XML-RPC has some glaring deficiencies in meeting the needs outlined above. This will be further elaborated when we specifically compare Web Services to XML-RPC later in this paper.

In February 2002, IBM, Microsoft, Intel, BEA and other companies formed the Web Services Interoperability Organization (WS-I), a non-profit organization for promoting Web services standards. "SOAP, WSDL and UDDI" was how Web Services have been described and thought of ever since.



Web Services – Business and Technical Reasons for Success

- The value proposition of Web Services is hard to resist: it's about interoperability, and the promise of reaching a big jump in capability with minimal changes and maximal re-use of existing infrastructure. Web Services empower businesses with more freedom to choose business partners and enterprise platforms and components.

- "Develop Once, Sell Everywhere" - Once a website/enterprise successfully implements an idea to become a success, the next level in extending their services and reach is to offer their core functionality as a Web Service so that other vendors and individuals can use their Web Services, thus enhancing their brand presence and in some cases, to also open up a new revenue stream, in addition to giving their business a competitive edge over their competitors.

- Web Services also have a significant role to play in the era of outsourcing professional services and focusing on core competencies. To provide the customer a seamless experience while accessing the services, the company is likely to resort to third-party Web Services, rather than out source the entire service.

- Web Services started off being a method for web systems to access remote legacy data. Surveys (META Group) indicate that more than 70% of critical data is still stored in legacy systems like IBM Mainframes, IBM iSeries, DEC VAX, etc. and decades-old applications implement 85% of the enterprise business logic and further that, the first wave of Web Services deployment will involve 90% of 'large' organizations employing web services to access these legacy systems over the next four years. For instance, among popular such Web Services, we have IBM's effort called Host On-Demand, which is the leader in this domain.

- Ever since its inception, some of the doubts raised about the feasibility and longevity of the Web Services technology have been dealt with in a positive manner by the Web Services community. For instance, Web Services seemed to compound the problem of measuring performance and availability of a distributed system by spanning across multiple operating systems, technologies, and enterprises. However, the well-defined structures and standards that characterize Web Service provides opportunities for performance and evaluation measurement that doesn't exist in other distributed environments.

- From a technical standpoint, Web Services are services that rely on a compact set of universally accepted standards (XML, SOAP, WSDL, UDDI) that offers a level of interoperability and platform independence unseen before in the industry. Since it is a message-based technology (as opposed to being API-based), different technologies and operating systems can communicate between each other using standard XML messages. Further, since it's a loosely-coupled architecture, if one service in an application is broken, it can effortlessly be substituted with a similar one, without impacting the rest of the application or taking the system down for maintenance.

Shortcomings about Web Services

The factors that will determine the success of Web Services technology start with their affinity across a variety of scales. Can the technology be used in the simplest grassroots project as well as the most complex business interchange? SOAP and WSDL pass this test well; UDDI, at least initially, is suited mostly to the higher end of the corporate Web Services. UDDI has not taken off in practice, and many have begun to question whether it ever will. When it does, it will not likely be with its original ambition.

The security aspect with Web Services is a big challenge because it requires interoperability across different technologies since users need to authenticate each time they cross a boundary; this is called the End-to-End credentials challenge. Web Services could do well with -

- A standardized mechanism to obtain security credentials.
- A standardized mechanism to represent these credentials in XML
- A method to package credentials into a SOAP message, so both client and server can decode and process.

Security Assertion Markup Language (SAML), WS-Security, XML Signature are some of solutions suggested by the industry to represent security credentials in XML and work with XML documents at the element level. Moreover, WS-Security is transport agnostic which means that it can function over any proprietary platform or system, protecting data at rest, in synchronous or asynchronous environment and providing end to end security across the life of a transaction.

Standards divergence – competing groups of vendors introduce proprietary technologies for security, management, reliable message delivery, etc. – the industry is still working on defining standards for these issues. In the absence of well-defined standards for the above issues, implementers are resorting to custom code, which defeats one of the strengths of Web Services - the tight structure imposed on it, in terms of technologies (XML, SOAP, WSDL, and UDDI). So the sooner the industry is able to arrive at a common standard and framework for the above issues, the better it is for the future of Web Services.

Management of Web Services – one may see the popular Service Level Agreements (SLA) be re-defined into more of a Business Level Agreement (BLA) to accommodate for the essential characteristics of Web Services.

Web Services vs. CORBA, COM/DCOM, XML-RPC

To better press a case for the bright future of web services, it has to be compared to the other similar, competing technologies.

Web Services vs. COM/DCOM

DCOM requires all participating nodes in a distributed application to be running a flavor of Windows and thus, is more proprietary in nature. Even if a different platform could be used, the support and documentation for other platforms is inadequate.

Web Services vs. CORBA

While one could dismiss the competition presented by DCOM somewhat by saying that DCOM is proprietary, thus negating the goal of standards-based interoperability, CORBA comes closer. However, some reasons why Web Services is likely to do better than CORBA are:

- Every node in the application would need to run the same Object Request Broker (ORB) product. However, CORBA ORB's from different vendors can inter-operate, but that

interoperability doesn't extend into higher-level services such as security and management, and vendor specific optimizations are also lost in the process.

- Secure interoperability in CORBA has always been a dream – while serious efforts are being made to enhance security features with Web Services, the industry still expresses skepticism as far as security with CORBA is concerned.
- CORBA-Interface Definition Language (IDL) is much less expressive than XML-schema - the latter makes integration in business/financial applications much easier
- CORBA clients – requires advanced knowledge of Java or C++.
- CORBA defines efficient binary protocol. SOAP is text-based and optionally includes type info as part of the message, which simplifies debugging and traffic monitoring since the message content is human-readable. CORBA IDL cannot accommodate attachments such as DOC or PDF files as part of the message, which SOAP allows MIME attachments as part of the message content.
- Microsoft never participated in the CORBA world, which has always been its political Achilles heel, while big heavyweights like Microsoft, IBM, Intel, and SAP are rallying right behind web services.

Web Services vs. XML-RPC

Currently, these 2 technologies are quite close in terms of their application. The Open-Source Community has promoted a XML-RPC approach to build web services as opposed to the SOAP standard. The essential difference between these 2 technologies comes down to their representation of messages and interaction. There are 2 styles of interaction – "document style" and RPC-style – document-style interaction involve fewer interactions and therefore richer data structures are passed through the interfaces. In RPC-style interactions, small amount of data are passed in multiple requests, synchronously receiving a small number of reply data.

A good analogy for illustrating RPC vs. document style is the difference between making a phone call and sending an email message. When making a phone call *with no voicemail available*, you are expecting some one on the other end to pick up and begin talking to you. This is very much a request-response paradigm and maps well to RPC-style messaging. On the other hand, sending an email doesn't require the sender to be there. The email, or business document, which has all the information it needs, can wait in a queue or mail server until the receiver wants to read it. This is analogous to the document style interaction.

The asynchronous model of interaction and the coarse-grained document-style is a more popular interaction mechanism today. While an RPC approach can be implemented rather quickly, it will not be sufficient down the road when you have to interact with suppliers, customers, and partners who are beyond your control. In these situations, you will want an architecture that shields you (and the clients) as much as possible from the back-end implementation.

Document style messaging delivers a number of benefits:

- With document style, you can utilize the full capabilities of XML to describe and validate a business document. With the RPC approach, the XML typically just describes the methods and parameters of the method call.
- It does not require a tight contract between the client and the service provider. RPC is typically static, requiring changes to the client when the method signature changes. With document style, the rules can be less rigid.
- Because it is self-contained, document style is typically better suited for asynchronous processing.

There is one key disadvantage to document style — it is typically more difficult to implement than an RPC approach. Despite this fact, the flexibility gains well outweigh the implementation costs.

Also, XML-RPC's bewildering insistence on ASCII strings alone (despite its using the well-internationalized XML) means it is really not suitable except in English-only contexts. It also has a rather haphazard choice of data types. In SOAP, procedures take named parameters and order is irrelevant. In XML-RPC order is relevant and parameters do not have names. But most importantly, XML-RPC is confined to simple request/response services with very uniform and highly structured message types.

.NET vs. J2EE

Both Microsoft's .NET and Sun's J2EE present themselves as good options to develop web services applications in. Here is a comparison between the 2 technologies:

.NET

- Built on industry standards like XML, SOAP, and HTTP
- Allows the use of multiple programming languages
- Delivers a lot of easy to assemble "out-of-the-box" functionality
- Provides a familiar GUI environment
- Restricted to MS Windows platform for the foreseeable future
- Developers are limited in terms of system design and data exchange models

J2EE

- Built on industry standards like XML, SOAP, and HTTP
- Cross-platform and cross-vendor
- Provides a wealth of choices for IDE's, application servers, XML parsers, SOAP engines, messaging platforms, etc.
- Allows developer flexibility in terms of design and data exchange models
- Development of web services is often complex
- Development is restricted to Java
- Flexibility can be overwhelming

Conclusion

The dynamics of commercial competition will certainly always be a major force in the shaping of Web services, but big commercial interests have tried forcing their way before in distributed computing, without the sorts of success and opportunities they find in the current era. This is because of the diversity that has braced Web services. The health of this diversity will continue to be the dominant indicator of the future of Web services technologies.

Web Services has been categorized as a '*disruptive technology*'. What makes a technology a disruptive technology is when a radically new technology disrupts the forces and trends shaping

the market by competing against an established technology that is well along the path of sustaining development. Other instances of disruptive technologies include mainframes and mini computers, professional workspaces and personal computers, and 3G wireless and Wi-Fi technologies. Typically, initially a disruptive technology cannot meet low-end requirements (Wi-Fi wasn't envisioned to create as many "Hot-Spots" as we have today), but as the technology evolves, it crosses the line bordering the high-end market, and begins to cause disruptive changes in the industry. However, the industry still believes that the moment of truth for web services hasn't arrived yet and the best of this technology is yet to unravel.

References

1. http://www.webservices.org
2. Web Services Journal Online
3. A Perceptive on Web Services; Enrique Castro-Leon; Intel
4. The Past, Present, and Future of Web Services; Uche Ogbuji; October 2002
5. Web Services at the Front End, Joe McKendrick
6. Getting Web Services Ready for Business; Alan Kotok
7. Applying Design Issues and Patterns in Web Services; Chris Peltz; January 2003
8. Management and Security in the World of Web Services, Dmitri Tcherevik, July 2003
9. Measurement Principles of Web Services; Jeffrey P. Buzen, Leo F. Parker; December 8, 2003
10. Is Web Services the reincarnation of CORBA? - Dan Gisolfi, July 2003
11. Integration approaches: Web services vs. distributed component models; Web Services Journal Feature; May 2003 by Sanjay Patil, Nick Simha
12. Integration approaches: Web services vs distributed component models: solutions for integration problems old and new. (Web Services Journal Feature); April, 2003, by Sanjay Patil, Nick Simha
13. CORBA and Web Services, Jorgen Thelin, Chief Scientist; PJ Murray, Product Manager; Cape Clear Software