**A Rational software whitepaper from IBM**
10/03



# Introducing Web Services into the Software Development Lifecycle

*Jeffrey Hammond*

Table of Contents

## The End of the Beginning….

Most people would agree that the Internet has fundamentally changed the way people relate to each other. In public and private life, people use the Internet to exchange thoughts, execute commerce, and maintain contact with each other in a way that transcends traditional boundaries of time and distance. The Internet has accelerated the formation of national and global "meeting places" as well as the creation of virtual markets where goods and services may be easily exchanged.

The explosion of technology and innovation that fueled the first phase of hyper-growth in the Internet community has largely passed. Massive expenditures on infrastructure and dubious business models have bowed to natural economic forces like profit/loss and the requirement to achieve a positive return on investment. As we step back and take a breath at the "end of the beginning" of the Internet revolution, it is only natural to examine what trends and technologies may lead the next wave of Internet growth.

One well-established business trend over the past few years is the use of the Internet and related technologies to automate execution of inter-party business processes. Companies like e-Bay and Amazon have proved that Internet-enabled inter-party business transactions are not only possible, but can entirely sustain a robust business model over the long term. These companies typify the first phase of the Internet's growth: B2C (Business-to-Consumer) business. B2C processes typically involve direct input from a user who queries a single business. The business provides information (product, availability, price) which the user employs to make a decision (place an order, make a reservation, terminate the process).

While user-initiated B2C business processes are common across the Internet, *automated inter-party business processes*, which do not require human interaction or decision making, are less typical. These processes are typically described as "B2B" or Business-to-Business processes. The B2B model is being driven by business' desire to reduce the role of non-automated intermediaries (people, paper forms, HTML, UIs) within the business process *when those intermediate roles or technologies do not add value.*

## Why Use the Internet for B2B?

The goal behind using the Internet to automate business processes is to reduce the costs typically encountered when executing automated inter-party business processes. For example, consider how many companies (and individuals) file tax information with Federal and State Tax Authorities.[1] Government agencies may offer EDI (Electronic Data Interchange)-based mechanisms to avoid re-keying of data, thus avoiding errors and significant duplicate entry of information. However, each EDI implementation is likely to be substantially different, requiring "one-off" development and additional costs. (Imagine having to implement the IRS standard to file your own taxes on-line!) As a result of proprietary standards and technologies, the cost is relatively high in terms of implementation and the resulting need for "toll-gate" fees to recover the costs of implementation.

At Rational, we believe that the ability of the Internet to lower the cost of automating inter-party business processes hinges on two fundamental cost drivers: *ubiquity* and *standards-based interoperability*. These cost drivers are the guiding force behind "Web Services" and they provide the basis for understanding how the new paradigm of Web Services can be put to best use.

## The Ubiquitous Web

The reality of today's business world is that robust access to the Internet is a *defacto* business requirement. Businesses must have an internet presence, and many employees use e-mail and Web access to communicate with each other. Even if all other access to the Internet is disabled or blocked, standard communication ports like PORT 80 (used for HTTP) and PORT 25 (SMTP) are likely to be in use. The ubiquity of these Internet standards and protocols has resulted in high demand, which has resulted in a rich supply of application and e-mail servers that form the backbone of today's Internet. This commoditization of Internet-enabled solutions has had the effect of significantly reducing the costs to businesses of having and maintaining an Internet presence (In a case where open source software like Apache is used, the software cost is driven to zero). This is in direct contrast to the traditional access mechanisms that are used to support execution of inter-party business

[1] For and example of this sort of EDI-based exchange of tax data and the assimilation effort, refer to http://www.boe.ca.gov/elecsrv/efiling/pdf/partII.pdf.

processes (e.g. EDI, dedicated access lines, proprietary networks, and specialized technology). The ubiquity of the Internet also has the effect of reducing the barriers to automation (cost, technology, set-up time). The premise of Web Services is that by building on ubiquitous Internet protocols and standards like TCP/IP, HTTP, and XML, a company can provide a lower-cost alternative to traditional access methods while maintaining a similar level of capability.

In many cases, the cost of maintaining the required infrastructure for Web Services is already viewed as a "sunk cost." As long as the additional "variable costs" for tooling, training, development, and execution of Web Services are less than the costs of maintaining traditional alternatives, then there will be a clear positive ROI for implementing a Web Services architecture. In addition, by using a common infrastructure that companies are already investing in and using on a daily basis, Web Service-based business processes can gain the same immediacy and flexibility that the Internet has brought to e-mail, searching, and shopping.

## Interoperating Standards

Anther important factor in reducing the costs of inter-party business processes is *interoperability*. Part of the reason that the Internet has proved so successful is that, by and large, anyone can use it from a variety of devices running different operating systems and applications. With detailed standard specifications like TCP/IP, FTP, HTTP, HTML, XML, and SMTP, implementation is separated from specification. For Example, HTTP can be used by any browser or other device capable of understanding the HTTP protocol specification.

Standards not only enable interoperability, but they also create an ecosystem where tools that enable automation can emerge and evolve according to market needs. Standards can also justify the cost of creating automated solutions around technology, because standards help collapse multiple markets into a single, larger market where larger investments become practical. In other words, as automated tooling appears to support a market, it improves the productivity of existing projects. A classic example of market formation around standards is the explosion of the RDBMS market as the SQL standard was accepted in the industry. As SQL became an industry standard, businesses had a variety of vendors to choose from, with a wide range of costs and capabilities. In addition, the SQL standard allowed Independent Solutions Vendors (ISVs) to introduce a wide variety of productive tools. Example of these tools include 4GLs like PowerBuilder, and Database Modeling tools like Rational Rose. In the end, the formation of a standard created a win-win-win situation for customers, businesses and ISVs alike.

An additional benefit of standards is that they allow for economic abstraction of the standard (sometimes creating derivative standards). This results in improved productivity, and can further enlarge a market by abstracting the problem to a higher level. The current availability of robust J2EE compatible IDEs like WebSphere Studio Application Developer, Borland JBuilder, and WebLogic Workshop is a good example of how standards promote the formation of higher level solutions. In this case, a larger audience can exploit the J2EE standard without needing to fully understand the internals.

## The Drive Toward Web Services Standards and Interoperability

Rational believes that interoperable, standardized Web Services are critical for both customers and solution providers to justify their investment. Without standards-based interoperability, it is likely that Web Services will suffer from the same integration difficulties and costs that have plagued the adoption of component-based development. For this reason, Rational is an active participant in the standards bodies that are working to eliminate incompatibilities between different Web Service implementations. An example of this is Rational's participation in the Web Services Interoperability organization (WS-I, at http://www.ws-i.org), which is an open industry effort chartered to promote Web Services interoperability across platforms, applications, and programming languages.[2] Rational also participates in other organizations focused on establishing the standards required to make Web Services successful. These include the W3C, Java Community Process, and the Web Services on WebSphere organizations.

WS-I has articulated the concept of "Profiles" which are named groups of Web Service specifications at specific version levels, along with conventions about how they work together.[3] The first profile to be standardized is **WS-Basic**. WS-Basic is comprised of four specifications:

[2] WS-I.org - http://www.ws-i.org/AboutUS.aspx.

[3] Web Service Profiles – An Introduction http://www.ws-i.org/docs/WS-I_Profiles.pdf.

- XML Schema 1.0
- SOAP 1.1
- WSDL 1.1
- UDDI 1.0

Rational fully endorses the WS-I Basic profile, and believes that it will help drive the interoperability required for Web Services to be successful. We believe that the release of WS-Basic and its support by WS-I members will create an important inflection point in the development of the Web Services ecosystem.

In the absence of industry wide standards, it is still possible to create interoperable Web Services, but only when the involved parties agree on what technologies to use. For example, a company could agree to only exchange simple data types which would allow basic interoperability between most Web Services. Alternatively, many companies are using Web Services to automate internal business processes. In this scenario, the company has control over their technology environment. This allows proprietary technology that provides additional capability to be deployed. In this way, issues like propagation of security and transaction context can be dealt with. The benefit to this approach is that a company can solve interoperability issues within their own firewall, and gain experience with Web Services as an early adopter. The risk is that a company may invest significant resources in non-standard implementations that will require re-factoring as interoperable standards are developed.

For more information about SOAP, WSDL and UDDI, refer to *Developing Applications with Web Services*[4]

## The Software Development Life Cycle

If the current players are successful in meeting the cost drivers defined above, then automated solutions for rapidly creating Web Services applications will be demanded by companies who are looking to implement or exploit Web Services capabilities. Companies will demand robust solutions that will address the needs of the entire software development lifecycle.

As software developers begin to introduce Web Services into the software development lifecycle, they will find that, at least initially, the challenges they face in implementing Web Services are actually an extension of the current challenges they encounter when practicing component-based development (CBD). This is because leading vendors like Microsoft and IBM, in order to support Web Services, are extending their component architectures (namely J2EE and .Net) with Web Services technology. Consequently, the implementation of any Web Services strategy based on .NET or J2EE will require deep familiarity with the underlying component architectures and their approach to component-based development.

While initial experiences with Web Services will require few changes to the software development lifecycle, over time Rational expects additional standards to emerge that will allow companies to build more powerful and sophisticated Web Services. Examples of problems that these standards will address include propagation of transaction context, security context (authorization, authentication and non-repudiation), and runtime change management support. These changes will require more specialized tools and processes to ensure developer productivity. It is also possible that, over time, Web Service implementation technology will evolve to be less dependent on component-based architectures (for example, consider a product that would directly wrap COBOL programs as Web Services). As standards and technologies evolve, solutions that automate the Web Services development lifecycle will need to adapt and become more specialized in order to provide tangible productivity benefits.

## Unique Challenges of Developing Web Services

While Web Services can currently be developed using the Tools and Processes already familiar to developers, there are challenges that must be met in order to assure success. The challenges are due to some unique characteristics of Web Services.

When designing applications based on a Web Services model, a company may find that it has less control over how pieces of a system are implemented. For example, if a business implements a service to provide weather

[4] Developing Applications with Web Services (Rational White Paper) by Jim Conallen and Dave Tropeano.

information, they have no control over the implementation technology that has been used by the provider. The Web Service consumer's insight into the Web Service stops at the WSDL interface definition. If the provider is using Microsoft .Net, and a security patch to Internet Information Server is released, the Web Service consumer may not know that the Web Service provider is shutting down the Web Service to apply a patch. Web Services are designed to be dynamically bound at runtime over the Internet. While this capability is extremely powerful, a Web Service creator may have difficulty identifying all the consumers of the Web Service they have produced. Unless procedures are put in place to identify all consumers of a Web Service, changing a Web Service implementation once it is deployed can be risky. In the example described above, the provider may have no mechanism in place to notify Web Service consumers that they need to take the Web Service off-line for maintenance. Interrupting service without notification may result in consumers switching to a more reliable Web Services provider.

Another challenge of Web Service development has to do with the nature of the Internet itself. While HTTP is truly ubiquitous, it is also stateless, and has no implicit quality of service. This means that Web Service design must deal with issues that HTTP and TCP/IP do not. Examples of these issues include security, transactionality, and state management. The paradox of Web Services is that HTTP is not a particularly good protocol to automate robust business processes. However the ubiquity of HTTP far outweighs the limitations of the protocol itself. While the additional capabilities Web Services require are layered over HTTP, the design challenge to businesses is not to push their Web Services design further than is practical.

Until standards are defined for implementing security, transactionality, and state management, business should focus on providing and consuming Web Services that are atomic, stateless, and have a low risk to information exposure. Examples of Web Services that meet these criteria are notification service like Stock Quotes, Weather, News Stories or Catalog or SKU services like Price, or Availability.

## Using Architecture Techniques to Design Web Services

Rational recommends several architectural techniques to business moving ahead with Web Services. Using these techniques can reduce the risks associated with introducing a new technology like Web Services into the software development process.

- Use UML Activity Diagrams to Model Business Process and Information Flow

  It is critical to identify what information needs to flow from producer to consumer. The earlier this information is identified, the easier it will be to develop a robust interface description. A good way to identify Web service information flows is to use UML *activity digrams* and *object flows*. *Swim lanes* can be used to indicate the boundaries between a producer and a consumer.[5]

- Practice Interface-Based Design

  Once an object flow has been described, a UML *interface* can be designed that properly represents that object flow. This interface is the basis for a specification of a Web Service Interface in *Web Services Description Language* (WSDL). After an interface has been designed, both consumers and producers can move forward with development. A producer is free to choose an appropriate implementation, or even change an implementation if the need arises. A consumer can create a lightweight simulation of the Web service for testing purposes, and proceed with application assembly.

- Design "Coarse Grained" Interfaces

  While the Internet has the capacity to serve massive amounts of information, each service request involves some amount of latency.[6] Often this latency can be a significant factor in overall response time. This model is perfectly appropriate for a document-centric architecture, which is typical on the Internet. This trait makes it appropriate to design interfaces that aggregate large amounts of information into a single request.

[5] Refer to p. 267 of *The Unified Modeling Language User Guide* by Booch, Rumbaugh, and Jacobsen (Addison Wesley) ISBN 0-201-57168-4.

[6] Latency refers to the time lag from the point a service is invoked to when it has returned a response to the client side consumer.

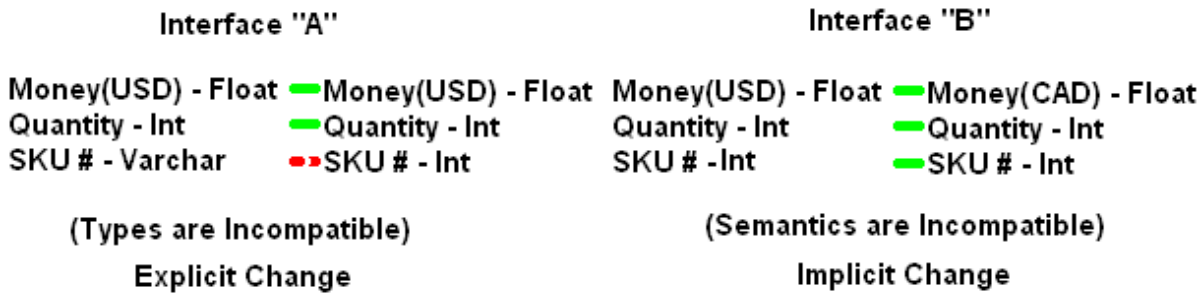
- Leverage Design Patterns to Ensure Scalability

  There are architectural design patterns that are well suited to implementing Web Services. By using these patterns, developers can optimize their system designs and provide Web Services that will scale as demand increases. Two examples of appropriate design patterns are *Information Leasing* and *Queued Implementation*. For more information about these patterns, refer to *Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications*[7].

## Additional Lifecycle Differences

The Web Services development lifecycle also differs from a CBD lifecycle in a few additional areas. These are described below:

The anonymous nature of Web Services makes runtime change management a critical issue. The primary risk of anonymous use is unintentionally breaking consumer integrations when upgrading a Web Service. This breakage could be *explicit*, where the WSDL interface and the SOAP messages change, or *semantic*, where the interface is unchanged, but the nature of the information being passed has changed.




The risk of runtime change management should be addressed through policy and tooling. An example of change management policy would be to only allow additive changes to interfaces. Another policy might be to deploy new versions of interfaces to a new URL location. A company might also implement *notification* in a Web Service design. For example, a previous version of a Web Service might continue to function, but it would return information that informs the consumer that the Web Service is deprecated, and will be taken off-line at a certain date.

As change management policies are put in place, they may be automated with lifecycle configuration management tools. For example, in the scenario described above, it is conceivable that a company may have several versions of a Web Service deployed at a single point in time. Each Web Service implementation needs to be maintained until consumers can migrate to a newer version of that service. This scenario can be easily supported through the use of parallel development streams.

Another issue is that until standards emerge for providing quality of service (QOS) data, aggressive testing of Web Services is the best way to ensure that a given Web Service is capable of meeting acceptable standards for scalability, reliability, and throughput. When a Web Service implementation is out of a company's control (as in the Weather service described above), testing can ensure that a Web Service solves the problems that are defined by the applications requirements. Test cases that use expected data can be used to verify that the information returned by a Web Service is technically and semantically correct. Test cases can be used to gauge the response time of a Web Service, and that the Web Service is up and running. Load testing can verify whether the Web Service can handle the load you expect to place on it. Finally, regression testing of deployed Web Services is helpful for insuring that a Web Service implementation remains consistent with the semantic data you expect it to provide.

[7] *Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications* (A Rational Whitepaper) by Alan Brown, Simon Johnston, and Kevin Kelly.

It is also important that producers of Web Services examine the operational characteristics of a Web Service in the targeted deployment environment. Since the producer has control of the Web Service implementation, the producer can use a testing tool to pinpoint performance bottlenecks and examine code coverage when Web Services are in use. This can help Web Service producers to make decisions about when to load balance, or re-factor their Web Services to optimize performance and reliability. Successful tool vendors will take into consideration both the consumer's and the producer's point of view as they develop testing products to automate the tasks required to effectively test Web Services.

## Initial Web Services Development and Beyond

With careful planning and application of the techniques that have been described above, it is possible to develop Web Services in a manner that mitigates risk and minimizes future rework. By establishing good architectural practices, and paying specific attention to change management and testing policies, early adopters can proceed and expect to deliver basic Web Service functionality within a business. At this stage of Web Services evolution, delivering B2B capable Web Services requires close cooperation between Web Service consumers and producers so that interoperability issues can be resolved.

As companies look to develop Web Services, they should closely watch the market to ensure that Web Services will truly be able to deliver on their promise of ubiquity and interoperability. While monitoring the progress of interoperable Web Services, here are some inflection points that may result in dramatically improved capabilities:

- Successful delivery of the WS-Basic profile and implementation by major vendors like Microsoft, IBM, and BEA.
- Successful delivery and implementation of a mechanism for providing an interoperable security context by major vendors. A developing standard is WS-Security, currently in development within the Organization for the Advancement of Structured Information Standards (OASIS).
- Successful delivery and implementation of a mechanism for providing transactional context between Web Services. A developing standard is WS-Transaction.
- Successful delivery of a standard way of executing workflow with Web Services. A developing standard is Business Process Execution Language for Web Services (BPEL4WS)
- Successful delivery of a standard for managing runtime change of Web Services. While some initial work is in progress, there is no clear standard on the immediate horizon.

As standards are delivered and implemented in each of these areas, Web Services will gain in capability and provide more compelling business value. One reason that Rational is working with IBM and Microsoft to advance these standards is that we think that .Net and WebSphere support of interoperable standards is critical to the success of the Web Services market. Success in these efforts will ensure that Web Services will play a critical role in the next phase of Internet development.

**IBM software integrated solutions**

IBM Rational supports a wealth of other offerings from IBM software. IBM software solutions can give you the power to achieve your priority business and IT goals.

- *DB2® software helps you leverage information with solutions for data enablement, data management, and data distribution.*

- *Lotus® software helps your staff be productive with solutions for authoring, managing, communicating, and sharing knowledge.*

- *Tivoli® software helps you manage the technology that runs your e-business infrastructure.*

- *WebSphere® software helps you extend your existing business-critical processes to the Web.*

- *Rational® software helps you improve your software development capability with tools, services, and best practices.*

**Rational software from IBM**

Rational software from IBM helps organizations create business value by improving their software development capability. The Rational software development platform integrates software engineering best practices, tools, and services. With it, organizations thrive in an on demand world by being more responsive, resilient, and focused. Rational's standards-based, cross-platform solution helps software development teams create and extend business applications, embedded systems and software products. Ninety-eight of the Fortune 100 rely on Rational tools to build better software, faster. Additional information is available at www.rational.com and www.therationaledge.com, the monthly e-zine for the Rational community.



IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Printed in the United States of America
01-03 All Rights Reserved. Made in the U.S.A.



The IBM home page on the Internet can be found at **ibm.com**

Part No. TP033A