

In this Issue

[SOA – What’s Slowing Your Business Down?](#)

[SOA – Old Technology that’s Slowing You Down](#)

[WS 101 – Web Services Standards](#)

[WS 102 – Standards that Your Team can Actually Use](#)

[WS 201 – Performance Gotchas](#)

[WS 301 - Interoperability Gotchas](#)

[WS 401 – New WS Capacities](#)

[WS 402 – Security](#)

[WS 701 – Taking it to the Next Level](#)

WebSphere SOA Performance Engineering



Tracking down tough performance issues across complex systems and applications is likely not an easy task. For new applications, it seems projects are always 98% complete for a long period of time. Often, the last 2% of the project includes the toughest part of the project that requires a specialized skill set - performance engineering. [Click here for more info](#)

High Performance SOA Part II - Built-in WAS SOA

In part I of this SOA (a.k.a., Service Oriented Architecture) series I discussed some of the **real business gotchas** related to SOA that if not covered off, can produce large failures for your organization and not to mention for your career. In part II of the SOA series, we primarily discuss the built-in Web services (a.k.a., WS) facilities available in the latest release of WAS version 6.1 (a.k.a., WebSphere Application Server). That is, functionality that is part of WAS ND but you may not know is there. The lineup for the SOA series of articles includes:

- **Part I – The Business of SOA**
- **Part II – Built-in WAS SOA – Web Services**
- **Part III – The ESB with WESB and WMB**
- **Part IV – Business Process Engineering - WebSphere Process**

WAS has both substantial but raw Web services capacity and a lightweight ESB (a.k.a., Enterprise Service Bus) in the form of a SIB (a.k.a., Service Interface Bus). This constitutes the basic components for a SOA solution. Incorporating the Process Server for process management and a heavier weight ESB via WESB and/or WMB will increase capacities in the form of higher performance, scalability and reliability. Incorporate a high performance “dashboard” so you know where you’re going. Considering ITCAM for RTT, SOA and WebSphere or equivalent is paramount. Lastly, there are a number of gotchas with these products as well as current standards that are addressed in this issue and follow on articles.

Change Fast – Neuroscience Tells Us How

According to new research, people who **actively learn** new ways rather than **are passively told** new ways tend to change their thinking faster. Active learning causes epiphanies that trigger structure neurological changes in the brain that are permanent.

The study indicates one part of the brain; the prefrontal cortex is responsible for change while the basal ganglia stores the familiar processes. The problem with change is that the prefrontal cortex takes a large amount of energy to sustain in terms of glucose supply. Therefore, people tend to use the old ways to conserve energy. The flip side of this is to call it “the lazy way”. It seems the study shows that learning at key points produces an **epiphany**. Using MRI’s, scientists have found that at the point of an epiphany new neurological structures are formed very rapidly in the brain that make the new way of thinking permanent.



The key points that come out of the neurological research that will maximize the change effort are:

1. **Prefer active learning by staff versus passive communication by leaders.** For example, if the company wants improved system reliability through automated monitoring, have people trained up on the products. At some point in the learning process, individuals will have that all important epiphany and your initiative will move along swiftly.
2. **Prefer physically fit people.** Fit individuals have a better supply of energy available for the key neurological change component – the prefrontal cortex.
3. **Prefer fast learners.** That means looking for evidence of fast learning in your people or future staff. Learning causes epiphanies and epiphanies cements in the brain the new ways of doing things.

SOA – What’s Slowing Your Business Down?

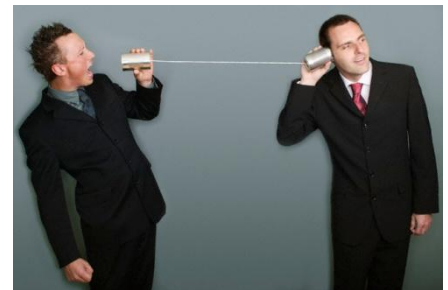


Research overwhelmingly has demonstrated that the fastest moving markets where a small edge can mean dramatic revenue gains include the finance, capital, distribution and insurance industries. What slows you down are processes that are not well understood that can neither work nor change fast. Compounding this is **expensive activities that are “baked in” with low value activities** that impede their re-allocation to either lower cost IT, lower cost labor or both. Think outsourcing low value activity, but only if you can

break the negative “baked in” approach. Currently, one of the best ways to “un-bake” your enterprise for high performance is through SOA. SOA process re-engineering success factors include tech savvy management, engineering versus just coding SOA, increased centralized SOA management and enterprise versus LOB based profitability incentives. The key question to ask is: Will LOB’s pay for the services or will access be free? Free access will likely result in over use of your services and associated resources. In addition, accounting for service charges will increase costs. **Bottom line is either you move faster or your customers will move to faster competitors.**

SOA – Old Technology that’s Slowing You Down

Even the **major technology vendors are frantically** replacing their old technology (that they use to sell to you) with their own new SOA based products – and they can’t do it fast enough to their own likely. Some of the biggest dogs are eating their own dog food - finally. Imagine what it’s like for your company. And the key technical reasons for the switch to the next generation of technology is the ability to move faster while lowering costs through **standards**, better ways to **discover** offered services in real-time and the reduction of a large problem that all enterprise encounter – **interoperability**. Arguably, the best framework for the next generation of technologies is SOA. One current downside of SOA is that technology standards can’t be agreed on and implemented in vendor products fast enough to keep up with the speed at which some markets need to change.



The key means to currently implement fast SOA is through Web services. The important new Web services capacities are global transaction capacity (V6.0), compensation on long running business processes (V6.1), message level security based on mature standard (V6.0), Stateful Web services and “publish and subscribe”



facilities (V6.1). **Interoperability** issues include WS-Security that does not work across WAS versions, the Software with Attachment (a.k.a., SwA) standard that is the accepted WS-I approach that Microsoft does not implement and complex data types that likely will not translate across platforms. **Performance** considerations include using Web services predominately close to the business domain, client caching for data with low update frequency and XML hardware accelerators for fast XML processing. Use static Web services for simplicity and low cost and dynamic Web services for enterprise flexibility. Lastly, for advanced needs, consider using an ESB as the broker in SOA solutions - either WebSphere Message Broker or WebSphere ESB.

SOA – A Quick Technical Review

SOA involves the orchestration of repeatable business tasks known as services. The top technical drivers for SOA are **standards**, the pervasive ability to **discover services** and widespread **interoperability** between systems and applications. Although SOA is technology neutral, the best technology currently available to implement it is Web services coupled with an ESB such as WebSphere Message Broker V6 and/or WebSphere ESB V6.

The core of Web services consists of a requestor, provider and a service broker as shown in Figure 1. There are two types of Web services – dynamic and static. Dynamic Web services have the provider publish services to the broker. When services are required, the requestor discovers the services by a request for service information from the broker. In the case of a *static* Web services, no service broker is involved and the addresses are “hard wired” into the requestor. This causes less desirable by albeit simpler coupling characteristics.

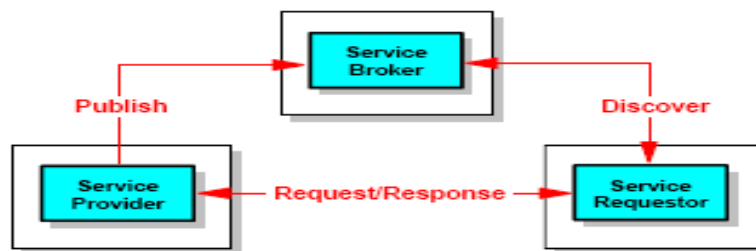


Figure 1 Service Oriented Architecture Fundamentals

In the context of WAS V6.1, the functions of the Service Broker can be accomplished by the built-in UDDI registry. Although not shown on this simple diagram, the built-in SIB could be used within WAS to mediate and transform requests between providers and requestors. For example, transformations could include changing a provider XML formatted message into a format the requestor understands. Lastly, Figure 2 shows the IBM products and where they fit within its reference architecture.



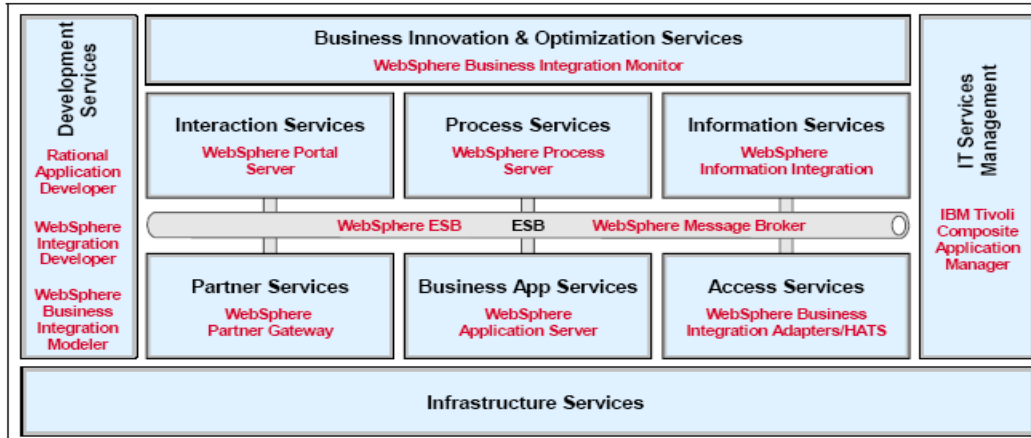


Figure 2 SOA Reference Architecture with Product Mapping

Web Services 101 – Web Services Standards that Really Matter



Web services is a software system designed to maximize interoperability between machines communicating over a network. It is also a set of specifications of which, only certain parts will likely be implemented into vendor products and used extensively by customers. In the long run, WS components that do not get used sufficiently will likely be deprecated from products. Figure 3 shows most of the Web services as of late 2006. The **most widely used and entrenched specifications** are the most mature (shown in blue) and include HTTP(S) (transport foundation), XML, SOAP, WSDL and BPEL (Business Process Execution Language). These standards provide the basics for communication and constructing services. The **up and coming WS**

standards include JMS, WS-Addressing, WS-Resources, WS-Security, WS-AtomicTransaction, WS-BusinessActivity. Here is a brief description of these key Web services standards:

- **SOAP** is the specification covering the exchange of XML based messages between the three actors in SOA – provider, requester and broker. Each soap message has a standardized format, envelope, header and body.
- **WS-Notification** standardizes the “publish and subscribe” (think newspapers) pattern. It now has been incorporated in WAS V6.1.
- **WS-Addressing and WS-Resources** together provide for Stateful Web services.
- **WS-Coordination and WS-AtomicTransactions** delivers the ability for Web services to engage in either single or two phase commit transactions (a.k.a., 2PC, global transactions).
- **WS-BusinessActivity** provides for the ability to engage in compensation for long running activities and thereby offers a vehicle for rollback to previous process states.
- **MTOM is an emerging standard for the transfer of attachment.** Currently, MTOM is a “vapour ware” standard that most vendors consider the future preferred means of document exchange such as PDF, Word and other document formats. The best current means of document transfer is either SOAP with Attachment (a.k.a., SwA) or pass by reference via a hyperlink to the document. Unfortunately, SwA does not interoperate with Microsoft .NET architecture.



- **WSDL** is an XML based document standard that details acceptable requests that a WS provider can make. The requests are in the form of operations, their parameters and data types passed in and out from the service provider. For dynamic Web services, the WSDL file is published to the service broker.
- **UDDI** – Universal Discovery and Definition Interface allows for dynamic Web service.
- **WS-Security** is an extension to SOAP that provides message level security. It includes message authentication, integrity through message signatures, and privacy through encryption.

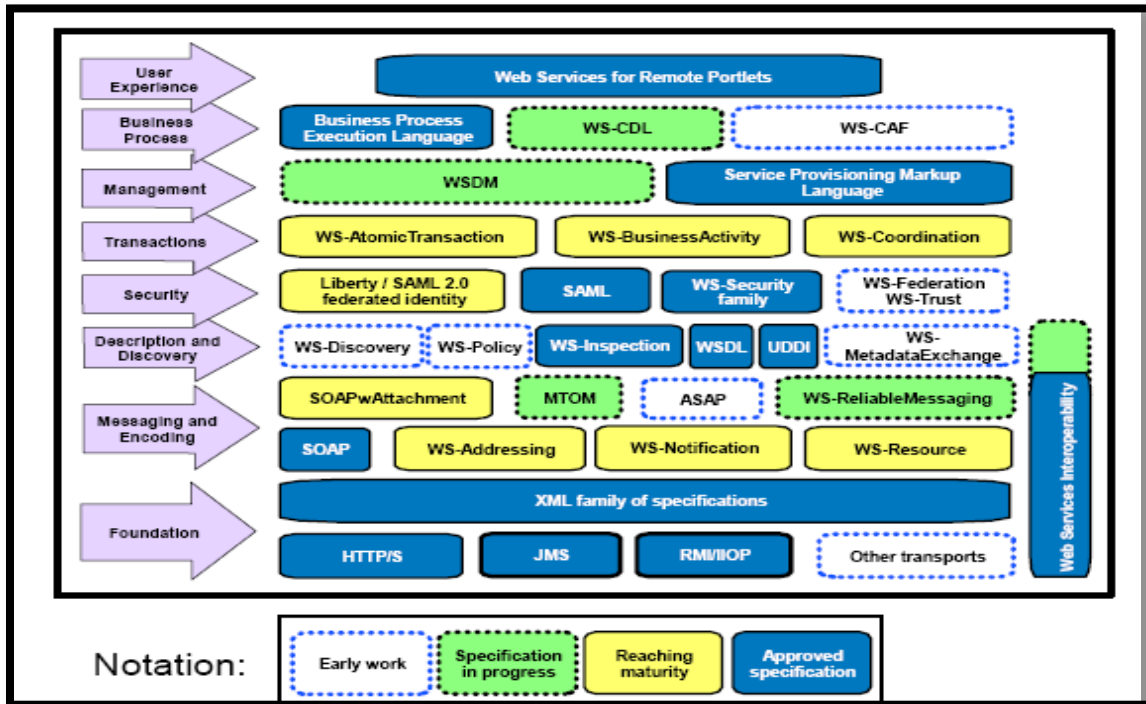
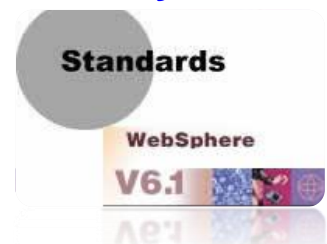


Figure 3 Web Service Standards Current Maturity

Web Services 102 – Standards that Your Team can Actually Use

The following table shows the standards added over the last several versions of WebSphere. The new additions in version 6.1 are the Stateful Web services ability through WS-Resources and WS-Addressing, publish and subscribe through WS-Notification and the ability to compensate for long running processes through WS-BusinessActivity. The early and stable standards include:



- **WS-I Basic Profile V1.1 (WS Interoperability Organization)** – includes SOAP V1.1, WSDL V1.1, UDDI V2.0 (Dynamic Web services) and XML V1.0 among some other minor standards.
- **JAX-RPC** – Java API for XML by RPC provides the protocol for distributed computing. Remote Procedure Calls (a.k.a., RPC) is a networking protocol that has existed for decades and almost every platform today has this as an operating system service.
- **JSR109 V1.1** – This standard is also known as WSEE V1.1 – Web services for J2EE.



- **Multiple Protocol/Encodes (SOAP/JMS, EJB)** - The WebSphere SOAP engine supports session EJBs and SOAP over JMS as Web services. This is an extension to the Apache Axis engine, which the WebSphere SOAP engine is based on.
- **Performance Monitoring** – There are many Web services related performance viewer counters including response time and payload size to name a few.

WebSphere Application Server	Web service runtime	Web service features
Version 4.0 Version 5.0	IBM SOAP (based on Apache SOAP)	Not WS-I compliant
Version 5.0.2 Version 5.1	IBM WebSphere Apache Axis V1.0 IBM SOAP	WS-I Basic Profile V1.0 JAX-RPC V1.0 JSR109 V1.0 SAAJ V1.1 UDDI V2.0 WS-Security (OASIS Draft 13) SOAP/JMS support Web services caching Web services performance monitoring
Version 6.0	IBM WebSphere Apache Axis V1.0 IBM SOAP (deprecated)	WS-I Basic Profile V1.1 JAX-RPC V1.1 JSR109 V1.1 SAAJ V1.2 UDDI V3.0 WS-Security V1.0 WS-Addressing WS-Coordination WS-AtomicTransactions JAXR support Multiple protocol/encodings (SOAP/JMS, EJB) Web services caching Web services performance monitoring
Version 6.1	IBM WebSphere Apache Axis V1.0	Same as V6.0, plus WS-Addressing WS-Resources WS-BusinessActivity WS-Notification

Table 1 WAS V4 to V6.1 Web Services Features

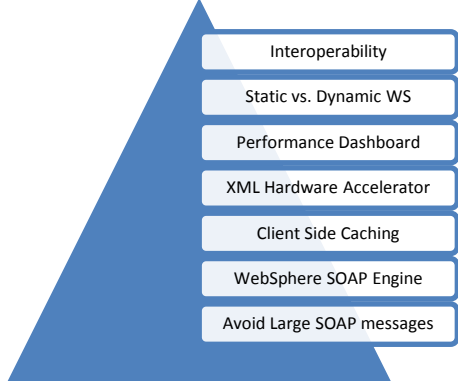
Web Services 201 – High Performance Gotchas that can Hurt You



Performance best practices includes implementing WS as close to the business domain as possible. Routinely use dynamic WS but for high performance, some services may have to be made static to increase throughput. Like a car in a Formula 1 race (you're in one if you didn't already know), consider providing your team with an intelligent dashboard so you can win at SOA – think ITCAM or equivalent. Also, consider a hardware XML accelerator, enabling client side caching, use the WebSphere SOAP runtime

engine and avoid large SOAP messages. The best understood and most deployed means of authentication with WS on WAS is using the LDAP token approach. Consider the following potential gotchas:



1. **Web services are designed for interoperability not performance.** There is a critical trade-off between performance and interoperability. SOAP over HTTP/HTTPS has a high cost related to the marshalling and un-marshalling of XML. Web services should be used where functionality needs are at a coarse grain level usually close to the business domain - as opposed to the technology domain. This decision will be more of an art than a science. For example, business versus technology domain would entail use cases such as “Get Checking” and “Encrypt Data” respectively. Messaging between applications that is typically high traffic is likely not a good candidate for Web services unless it solves a large interoperability problem - the “density” of the message is likely to be too low. This will slow throughput while dramatically increasing resource utilization.
 
2. **Use static Web services to keep things easy, simple and lower cost, use dynamic Web services if you need the flexibility and your team can handle the additional complexity.** When a Web service will not likely change much in the future in terms of location, request/response format or data types, embedding destination addresses in the applications (called endpoints) makes sense. When either the opposite is true or you want to keep the flexibility in the system, use dynamic Web services. This will require services to be registered in a third component called the broker. The cost and complexity increases. The headaches may increase as well.
3. **Incorporate ITCAM for WebSphere, RTT and SOA or equivalent tooling to improve performance and lowering resource costs while improving reliability.** Cars don’t come without dashboards and there’s a good reason – you won’t be able to see how you and your car are doing – out of oil or out of gas in the middle of nowhere or in the middle of a race isn’t an option. Likewise, SOA and WebSphere can get you anywhere you want to go, you just better be able to know how you’re doing and how to fix it and fast. ITCAM for SOA, RTT and WebSphere will provide you with the performance and reliability dashboard you need while making you and your team look like performers.
4. **Consider an XML hardware accelerator such as the IBM DataPower product.** Various models can act like a lightweight ESB with capacity that includes context routing, XML transformation, schema validation and protocol switching (between, HTTP(S), MQ, FTP and others). Offload XML to HTML transformation activity that is done in software by WAS or Portal to the DataPower hardware. Also, requests with XML can be pre-validated before being sent deeper into internal tiers that are much more costly, mission critical resources. Validation can include security inspection of XML for such things as DoS (a.k.a., Denial of Service) attacks. The appliances can also interoperate with TAM (a.k.a., Tivoli Access Manager) and enforce authorization policies early in a requests lifecycle that will pre-empt poor use of critical resources.
5. **Enable client side caching for non-sensitive Web services data.** Sensitive data should not be cached for high levels of security. Cached data should be predominately read only with a requirement for low frequency of updates to prevent the risk of using stale data.
6. **Use the WebSphere SOAP runtime.** This is the only engine that IBM will support. It also has extensions for JMS over SOAP and session EJBs as Web services.



7. **Avoid extremely large SOAP messages.** When designing your Web services, try not to send very large SOAP messages (for example, more than one megabyte). It is important to remember that a large percentage of processing time is spent in just the parsing of the SOAP messages. Therefore, large SOAP messages will cause a great amount of parsing. This will result in high processing loads, and low throughput. Also, avoid sending large chunks of binary data within the SOAP message. Java byte arrays (byte[]) can either be mapped to xsd:base64Binary or xsd:hexBinary. In both of these cases, the raw binary data must be converted into another format (base64 or hexadecimal) that takes up more space. Moreover, there is the added performance penalty of converting a byte array into the XSD format, and from the XSD format to the byte array. SOAP with attachments (SwA) may be an alternative. However, SOAP with attachments is not interoperable with .NET Web services.
8. **Use the Web services TCP/IP Monitor** – This tool will help expedite problem determination and assist in eliminating bottlenecks in WS performance.

Web Services 301 – Interoperability Gotchas that can Hurt You



The key standardization governance body is WS-I which both IBM and Microsoft are two of the leading members. Reducing gotchas include deploying WS-I compliant applications, checking all systems for expected WS standards, using WS-Security across WAS V6 or above only, minimizing use of attachments (for now) and reducing the use of complex data types. The important grouping of standards (a.k.a., profiles) in terms of

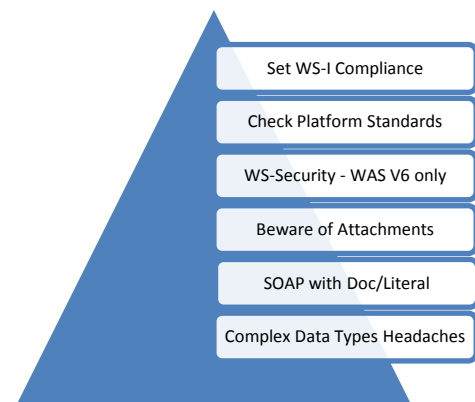
interoperability includes:

- **WS-I Basic Profile V1.1**
- **Attachment Profile V1.0** – Guidelines for attaching documents to SOAP messages.
- **Basic Security Profile** – WS-Security with REL, Kerberos, SAML, X.509.

Key interoperability gotchas include:

1. **Deploying Web services that are fully WS-I compliant.** This does not guarantee interoperability because other services may not be compliant. However, it is a good start. Set compliance in RSA/RAD 6/7 to “required”.
2. **Check expected standard versions across vendor versions and platforms.** If interoperating with other platforms (i.e. Microsoft) or earlier versions of WAS, make sure what versions of the standards they are expecting. For example, are the versions of Microsoft Windows Server expecting SOAP V1.1 or SOAP V1.0.

Gotchas – your team spends a month troubleshooting a problem only to find out the Microsoft platform had a service that was expecting another version of XML.



3. **WS Security across WAS versions will not work.** WAS V5 and V6 will not interoperate with WS-Security. WAS V5 was based on a WS-Security draft while V6 is based on V1.0 of the specification.
Workaround – None. Update WAS V4/5 applications to V6.
4. **Sending attachments is not going to be easy with the current state of standards.**
 - a. **Microsoft does not support SwA even though SwA is currently the WS-I standard.** Also, XML has no efficient solution for binary data. The WSC XML schema specifies binary data should be base64. This balloons code by 50% versus non-encoded data.
 - b. **SOAP with Attachments (SwA) and WS-Security do not work together.**
 - c. **MTOM is the standards that vendors are championing but it's currently "vapourware".**
Workaround – Transfer attachments out of band. For example, send a URI where the attachment can be retrieved.
5. **Use SOAP with Document/Literal.** There are several combinations of communication and encoding styles but for the best interoperability across J2EE and Microsoft platforms is **document/literal**. SOAP is covered under the WS-I Profile V1.1, which states the SOAP V1.1 specification should be used.
Gotchas – It's a message oriented approach and requires more programming work than the best alternate, RPC/literal.
6. **Complex data types will not likely translate across platforms with different architectures (.NET vs. Java).** In addition, beware of bottom up Web services design that may have complex data types. Using Collections in Java and attempting to translate to .NET types will likely not work correctly.
Workaround - Stick with built-in types such as Strings and Integer to provide greater interoperability. Carefully examine the types that are produced in the WSDL. Refactor to include only simple data types such as String and Integer since most platforms have these types.

Web Services 401 - New WS Capacities that Solve Your Problems



WAS V6 has several new WS capacities including global transactions, compensation on long running processes, message level security, Stateful WS and publish and subscribe capacity. Only the global transaction capacity has low work effort and a small learning curve.

1. **Global Transaction Capacity (a.k.a., 2PC)**
 - a. **Why it's important: Guards application data integrity** - Almost every corporation has multiple datastores and many applications have to manipulate data across several sources. This requires two-phase commits that almost every realistic scenario within large corporations has today.
 - b. **How it's done: Deployment time (Declarative), Low Work Effort.** Neither WSEE standards, J2EE 1.5 nor WAS 6.1 allows a web service to participate in the global transaction. However, the combination of WS-Coordination and WS-AtomicTransactions provide this capacity through issuance of a WS-AT ContextCoordination. This is simple to configure at deployment time through the deployment descriptor - it involves a configuration switch being applied versus additional code injection written by developers. Therefore, applications do not have to register



explicitly WS-AT participation. The WAS transaction services coordinates the transaction. Thus, a transaction that needs to gather information from several sources can have any mixture of JDBC, JCA or Web services calls that constitutes a global transaction.

- c. **Potential Gotcha's – Intra-Enterprise only** - Not recommended for across enterprise domains, that is transactions across enterprises.

2. Compensation on Long Running Processes

- a. **Why it's important: To guard process and application data integrity** - Organizations engaging in business process automation need to be able to rollback process state to a previous points-in-time. This while maintaining integrity of the process and corporate data. For example, a mortgage application is about to complete on Thursday when an error is discovered in property valuation that occurred much earlier in the process. The process may have to be rolled back, process state updated and database changes undone. This requires compensation. On long running process activities this is possible but only if WS-BusinessActivity is enabled. These types of activities include steps in a process that each run in the range from minutes to days; maybe even weeks whereas short running ACID transactions listed in #1 typically run sub-second durations.
- b. **How it's done: Programmatic, High Work Effort** - The WS-BusinessActivity (new in WAS V6.1) standard and implementation in WAS provides the basis for the compensation services. Compensation services are off by default and must be enabled for EJB's that need to participate in compensation.
- c. **Potential Gotcha's – The benefit can be large but there is a complexity cost.** Compensation services depend of significant additional engineering, design and development work in the form of matching compensation beans and data objects (SDO – Service Data Objects). The compensation beans and SDOs are used to retain integrity while rolling back to a previous point-in-time along the associated business process.

3. Message Level Security

- a. **Why it's important – Tighter Security, Higher Performance** - The big advantage of WS-Security is that it will allow security to be placed on different parts of a SOAP message. This lends itself to the “publish and subscribe” pattern where there is one provider but many interested parties. For example, the ability to send one large message out to several destinations with several sections secured. Vendors bidding on one or more parts of a reservation for a hotel, car and air travel often works this way. A message may be composed of several pieces of information where each piece should only be understandable by the intended receiver. That is, message part 1 is understood only by receiver 1 and message part 2 should only be understood by receiver 2. However, each receiver receives the entire message so there needs to be a means to secure selectively each part of the message. In this case, information security has to be done at the message level versus transport level (e.g., HTTPS).
- b. **How it's done: Programmatic, High Work Effort.** WS-Security code is embedded in the SOAP messages.



- c. **Potential Gotcha – Higher Complexity** - Message level coding adds an additional level of complexity that may be too sophisticated for the organization's skill level. Security defects go up as the complexity of the software increases. Complexity of testing also increases. This all adds costs that must be weighed against benefits. The alternative is to push security to the transport layer with deployment time work effort only.

4. Stateful Web Services

- a. **Why it's important – Higher Performance, Improved Efficiency** - Situation may arise where several decisions need to be made across multiple WS calls to finalize an answer from a service. In that case, it may be easier to have the service hold the state information instead of marshalling intermediate data back and forth. The retaining state problem was first seen in J2EE between the Web browser client and the server.
- b. **How it's done – Programmatic, Medium Work Effort.** Addition of WS-Addressing and WS-Resource code to Web service requests so that the provider service will hold state related information across several requests.
- c. **Potential Gotcha's – Service Reliability** - Now your requests are dependent on the reliability of the service provider. That is a risk that you may not be able to control. An ESB in coordination with the service provider must be configured for failover where the alternate services have access to the state. A less troublesome issue is additional provider resource consumption mostly in the form of memory utilization. This could be a problem if there is a lot of demand for the service. Additional service management is required in terms of the maximum resource pooling and service lease times that provider's assign to requestors. Similar issues were encountered and resolved at the web client to J2EE server boundary via HTTP session and clustering features first available in WAS V3.01.

5. Publish and Subscribe Capacity

- a. **Why it's important – Scalability, Create information once, distribute too many** – The “publish and subscribe” pattern is a powerful paradigm, which has been extensively considered in standards such as J2EE and in products such as WebSphere Message Broker (a.k.a., WMB). Both J2EE and the WMB have existed for many years. Many corporations need the ability to produce a piece of information and distribute it to many parties of interest. The important characteristic is scalability, create once and distribute it to many subscribers.
- b. **How it's done – Programmatic, High Work Effort** – The WS-Notification 1.3 specification level is used to implement publish and subscribe behavior. WS-Notification depends on WAS's built-in SIB (Service Interface Bus).
- c. **Potential Gotcha's – SIB scalability** - Not many infrastructure teams have a lot of experience with the SIB component of WAS.



Web Services 402 – Security

Broad infrastructure security for WAS and Web services is likely to be assisted by external components such as ITAM (a.k.a., IBM Tivoli Access Manager) and a LDAP directory product. The best mechanism for authentication is to use LDAP tokens. Tokens attach to requests and propagate (typically vertically) through components to minimize additional authentication queries to the LDAP Server. This is the most widely used approach. Web services security specifically included in WAS V6.1 consists of:

1. WSEE Version 1.1 - Web services for J2EE
2. WS-Security 2004



WSEE – Web Services for J2EE

WSEE defines the required architecture for Web services for the Java 2 Enterprise Edition (J2EE) environment. The packaging, deployment, and programming model for Web services in a J2EE environment are standardized with WSEE. WSEE-compliant services are portable and interoperable across different application server platforms. WebSphere Application Server Version 6.1 supports WSEE Version 1.1. The compliance to WSEE is a defined requirement in the J2EE 1.4 specification. Although WSEE does not restrict any implementation, it only defines two mechanisms to create Web services:

1. A stateless session EJB in an EJB container
2. A Java class running in a Web container

There are three key issues with WSEE:

1. **Web services cannot participate in global transaction (2 Phase Commit) with just WSEE mechanisms.** An existing transaction is suspended before a Web services port is accessed. WS-AtomicTransaction must be enabled.
2. **WSEE uses basic security mechanisms built into WebSphere.** WSEE is not based on WS-Security specifications. WSEE largely depends on the J2EE specification:
 - a. Authentication – Basic (user/password) or symmetric HTTPS using digital certifications.
 - b. Authorization - Secure the HTTP POST access of JAX-RPC service endpoints.
 - c. Integrity and Confidentiality - The only alternative is to secure communications by using SSL.
3. **Web services with WSEE don't provide for state persistence.**

How WSEE is implemented in WAS

There are two protocols that deliver Web services within WAS - either SOAP over HTTP or SOAP over JMS. Shown in the figure below for SOAP over HTTP, a request in the client is made to the Java Proxy that transforms it to SOAP over HTTP request. On the server, the request is processed by the WebSphere SOAP engine within the web container. The engine calls the java bean as a Servlet. If the Web service is developed as an EJB, the engine directs it through a router component. For the case of SOAP over JMS, a JMS Sender is required on the sender side. An EJB MDB listener is required on the server side.



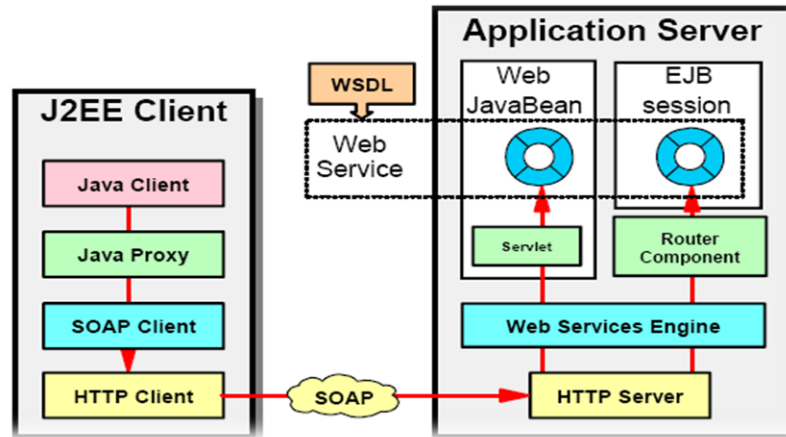


Figure 4 SOAP over HTTP

WS-Security

The figure below shows how WS-Security provides improved security when intermediaries are involved. In this case, HTTPS transport may not protect information across the intermediary. WS-Security should be used in the following situations:

1. **Multiple parts of a message need to be secured differently.** For example, a soap message has credit card and debit card information and each part must be private to different vendors. The point is, to stop both from seeing the other’s information - each needs separate means of privacy.
2. **Intermediaries are used so security needs to be with the message, not just the pipes.**
3. **Non-HTTP transport protocol is used.** WS-Security is transport protocol independent and works even when protocols are switched.



Figure 5 Web Service client to server security

WS-Security can be applied at the message versus the transport level as shown in Figure 6. Authentication, integrity and confidentiality can all be implemented at the message level or the transport level.



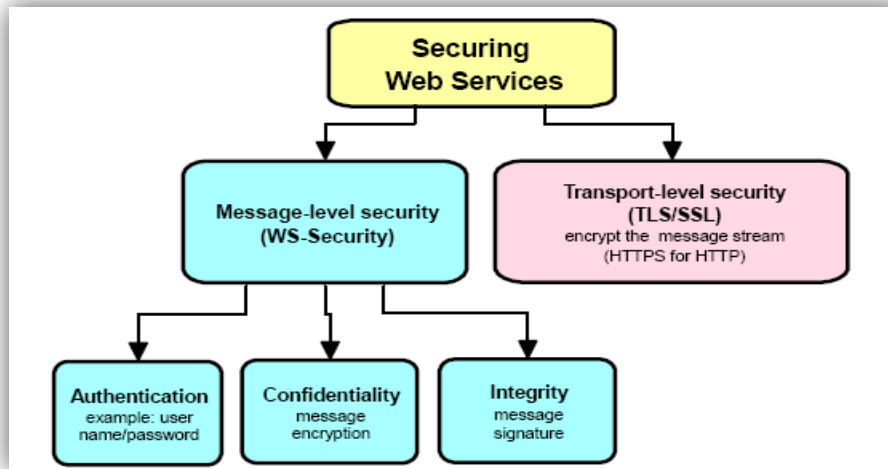


Figure 6 Message (WS-Security) versus Transport Layer Security

Figure 7 shows the SOAP message structure including the WS-Security related header. This area will typically include the tokens required to decrypt the data present in the SOAP body.

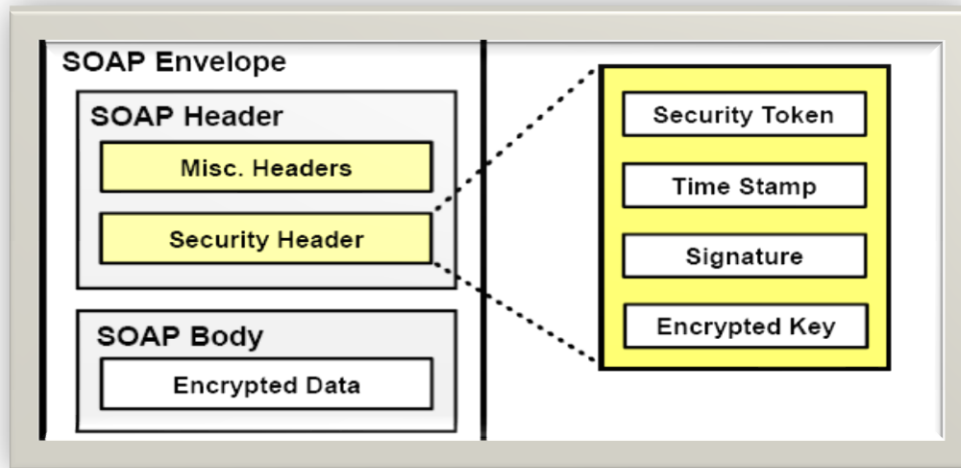


Figure 7 SOAP Messaging including WS-Services

Integrity and Confidentiality

Integrity is provided by applying a digital signature to a SOAP message. Confidentiality is applied by SOAP message encryption. In WAS version 6.1, multiple signatures and encryptions are supported. Essentially WS-Security allows message encryption and message signatures on various parts of the message using multiple certificates and keys as shown in the figure below.



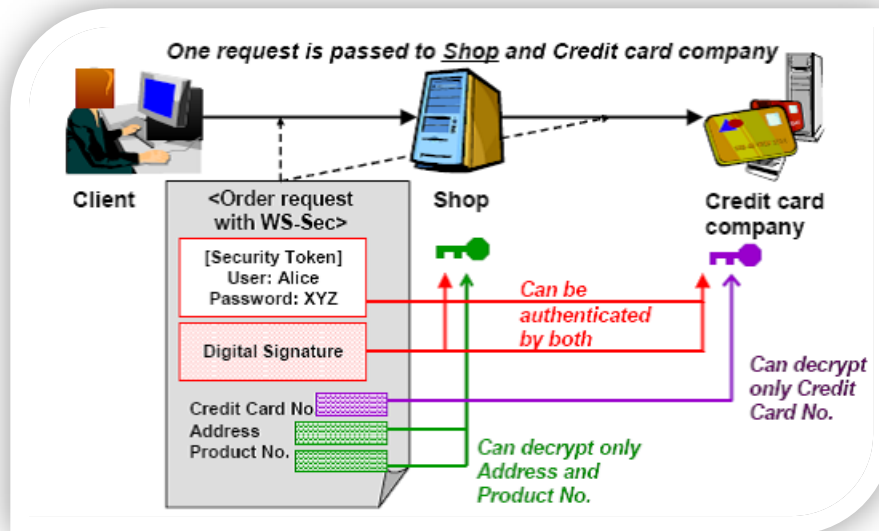


Figure 8 SOAP Message with WS-Security example

Using WS-Security, message confidentiality and integrity can be enforced for multiple receivers within one SOAP message. This is one of the most important advantages of WS-Security over SSL, because with SSL, it is impossible to protect message security for multiple receivers.

Web Services 701 - Graduate School - Taking it to the Next Level



Consider adding an **ESB** (a.k.a., Enterprise Service Bus) to SOA/Web services implementations because it upholds the spirit of SOA. This includes characteristics of loose coupling and easier component reuse. ESB's core abilities include message routing, protocol switching and in transit message transformations capacities. ESB's work well in situations where there are large numbers of system that typically create a "rat's nest" of point-to-point connections. A key benefit is application network addresses are moved out of

the applications and stored in the ESB. This approach hides and manages the endpoints for the application. Also, organizations can rapidly develop message transformations that can be applied to in-transit messages - for example, COBOL copybook to XML can easily be mapped in one-tenth the time it takes for developers to do it manually. Additional instances of services can be added to increase availability while making maintenance more flexible with improved modifiability. Both IBM WebSphere ESB and WebSphere Message Broker V6 (Advanced ESB) are options that should be considered as shown in Figure 9.



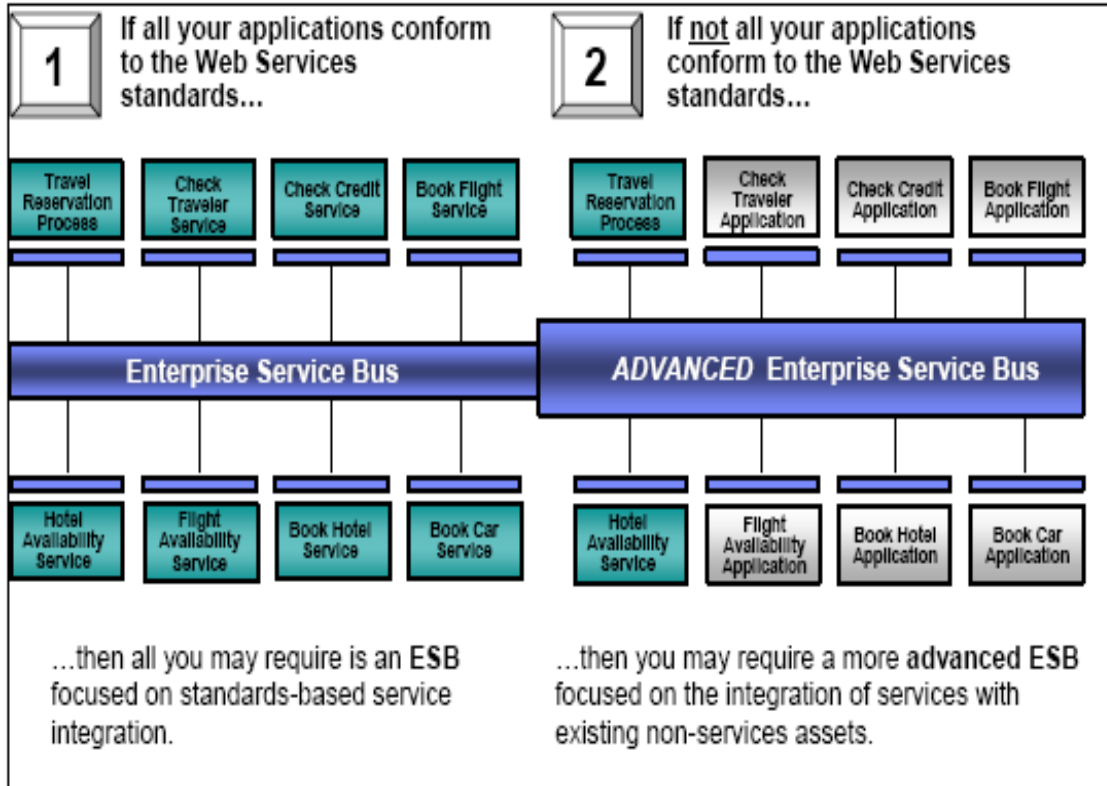


Figure 9 ESB Options - WMB or WESB

