Secure, Reliable, Transacted Web Services

Architecture and Composition

Level: Intermediate

Donald F. Ferguson, IBM Fellow and Chairman IBM Software Group Architecture Board, IBM Corporation
Tony Storey, IBM Fellow, IBM Corporation
Brad Lovering, Distinguished Engineer, Microsoft Corporation
John Shewchuk, Web Services Architect, Microsoft Corporation

October 28, 2003

Introduction

Today Web services - specifically distributed services that process XML-encoded SOAP messages, sent over HTTP, and described using Web Services Description Language (WSDL) - are being deployed broadly.(See "XML, SOAP, and HTTP" sidebar) Web services are used in a range of application integration scenarios: from simple, ad hoc, behind-the-firewall, data sharing to very large-scale Internet retailing and currency trading. And increasingly Web services are being applied in mobile, device, and grid scenarios.

Web services provide interoperability between software components that can communicate between different companies and can reside on different infrastructures. This solves one of the most critical problems that customers, software developers, and partners face. HTTP and SOAP provide communication and message interoperability. WSDL provides the description of the service to support interoperability between development tools; it complements communication interoperability with the ability to exchange interface definitions.

The basic set of Web service specifications enables customers and software vendors to solve important problems. Building on their success, many developers and companies are ready to tackle more difficult problems with Web service technology. The very success of Web services has led developers to desire even more capabilities from Web services. Since meaningful tool and communication interoperability has been successful, developers now expect the enhanced functions to interoperate.

In addition to basic message interoperability and interface exchange, developers increasingly require that higher-level application services interoperate. Many commercial applications execute in an environment ("middleware" or "operating systems") that provide support for functions like security and transactions.

IBM, Microsoft, and others in the industry are often asked to make Web services more secure, more reliable, and better able to support transactions. In addition we are asked to provide these capabilities while retaining the essential simplicity and interoperability found in Web services today.

This paper provides a succinct overview for the set of Web service specifications that address these needs. For the details of the specifications we provide references to the actual documents. The main purpose of this paper is to briefly define the value these specifications provide to our customers. We also describe how these specifications complement each other to compose robust environments for distributed applications.

We face a key engineering challenge: How do we give Web services new security, reliability, and transaction capabilities

without adding more complexity than needed?

Composable Services

As we've done with SOAP and WSDL, IBM, Microsoft, and our partners have followed the design principle of *composability* in the definition of Web service specifications. The approach we have followed is based on the design principle of *composability* in the definition of Web service specifications. Each specification we developed solves an immediate need and is valuable in its own right. For example, developers can adopt reliable messaging to simplify their solution development or adopt BPEL4WS to define their service compositions. And while each specification stands on its own, they are designed to be combined and work with each other.

XML, SOAP, and HTTP
The terms XML, SOAP, and HTTP are in common use today and in many respects their use has moved beyond their original acronyms. For completeness these acronyms are listed here: XML - eXtensible Markup Language, SOAP - Simple Object Access Protocol, and HTTP - HyperText Transfer Protocol.

We use the term *composability* to describe independent specifications that can be combined to provide more powerful capabilities. Operating system and middleware providers can support composed capabilities, e.g. providers can integrate reliable messaging support for communicating BPEL4WS processes. This example combines two independent specifications to simplify the development of communicating processes by eliminating the need to handle message communication errors during process design.

Composability enables *incremental consumption* or *progressive discovery* of new concepts, tools and services. Developers only need to learn and implement what is necessary, and no more. The complexity of the solution increases only because the problem's requirements increase, and is not due to technology "bloat."

Composability has and continues to be one of the key design goals for Web services. Over the last several years we have defined the most basic Web service specifications (SOAP and WSDL) to inherently support composition. One of the fundamental characteristics of a Web service is a regular, multi-part message structure. This structure enables the *composition* of new functionality. New message elements supporting new services may be added to messages in a manner that does not alter the processing of existing functionality. For example, it is possible to independently add transaction identifiers and reliable messaging sequence numbers. The two extensions do not conflict with each other and are compatible with pre-existing message structures.

Figure 1. Composability allows for using elements on an as-needed basis.

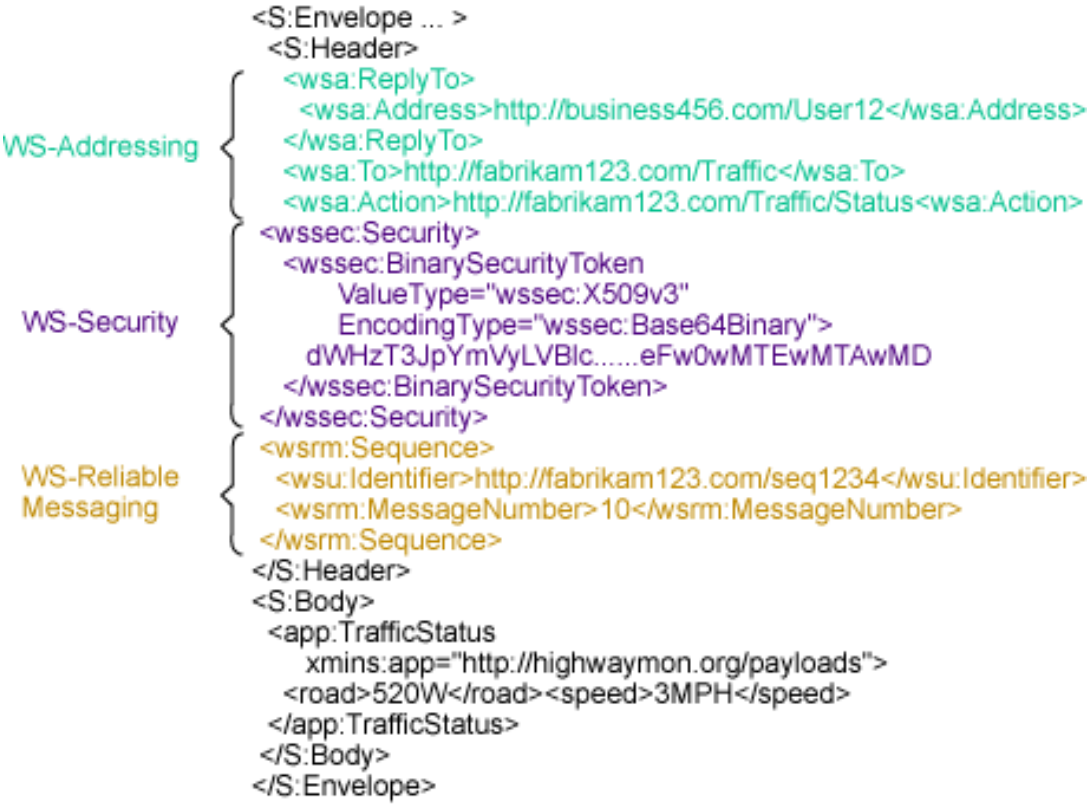


Figure 1 shows a simple Web services message that contains elements for WS-Addressing, WS-Security, and WS-

ReliableMessaging. Notice that the WS-Addressing, WS-Security and WS-ReliableMessaging elements are independent and these elements can be used independently without altering the processing of other elements.

This characteristic enable security, reliability, and transactions to be defined in terms of *composable* message elements.

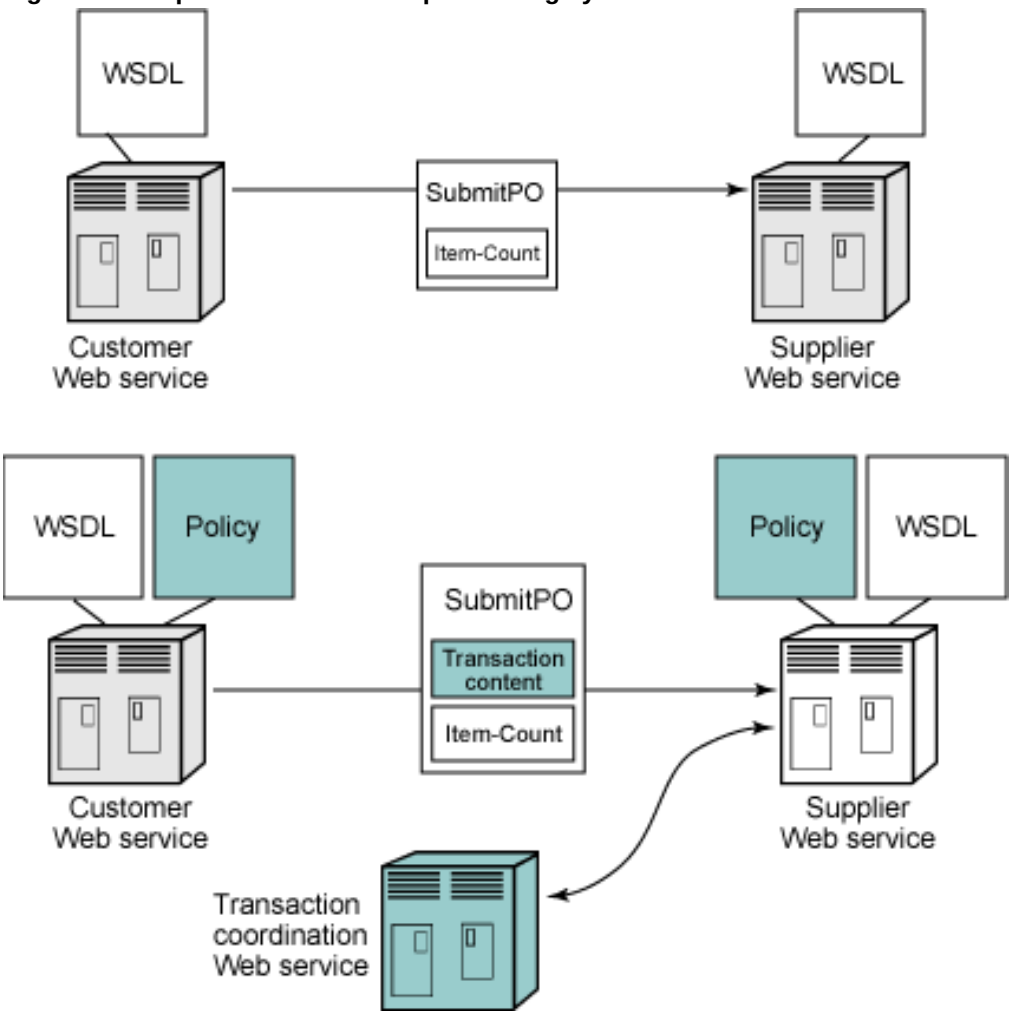
The notion of composition also allows for the creation of a specific set of well-defined composable Web services that support security, reliability, etc. These well-defined services specify the behavior of services necessary to support higher-level Web service functionality. An example is the Secure Token Service defined in WS-Trust that issues and validates security elements in messages.

Moreover, it is important that consumers of a service be able to determine the supported and required service assurances. The services must document their requirements and support for transactions, security, reliable messaging, etc. WS-Policy enables Web services to incrementally augment their WSDL to document what transaction, security and reliability functions they support or require. WSDL and WS-Policy enable composition for the description of services. This in turn enables the other parties to understand what message elements and higher-level services to employ when interacting with the service.

An Example of Composition in Practice

Figure 2 provides an overview that shows composition in practice. A customer and a supplier use Web services to process orders.

Figure 2. Composition in an order processing system



In building these Web services the developers use WSDL and related documents to describe their business interface. These WSDL documents describe the set of messages the customer and supplier Web services will process, e.g. the SubmitPurchaseOrder (SubmitPO) message that flows from the customer to the supplier. This is shown in the top of Figure 2. Once the core pieces of the application are in place, the developers can then decide to support an additional capability, for example, here they decide to make the order processing transacted. To do this they compose the following elements into the

existing structure:

- The services associate WS-Policy documents describing their support for transactions to their WSDL description of their services. Note that these policy statements augment but do not fundamentally alter the existing business functionality.
- To support transacted processing, the services add an additional message element describing the transactions that composes with, but does not fundamentally alter, the existing application messages.
- When the supplier service receives messages that contain the transaction element, it uses this information to communicate with a designated Web service called a coordinator that supports the transaction function. Again, this additional Web service merely adds to the solution and does not require modification to the description of the existing business functionality.
- Finally, the services may implement additional operations to support integration with the transaction coordinator service.

In the preceding figure these additional elements are highlighted.

The model is incremental and composable because:

- Adding the new functions can be done independently of adding other functions.
- Adding the function does not disrupt the existing messages, message processing logic or WSDL.

Composability is an increasingly important design principle, yet the approach is not always well understood. While the individual WS specifications define how individual elements and services interoperate, this white paper is intended to provide an overview of how the collection of specifications can be composed to provide more sophisticated interoperable Web services.

Web Services: A Service-Oriented Architecture

In recent years we have witnessed a flurry of activity centered on Web services development. With all of this activity it is important to step back and ask the question, "Why?" Certainly, Web services don't enable new kinds of computational capability - after all Web services still run on existing computers, executing the same set of instructions and accessing the same data. Moreover, Web service protocols in many cases can actually increase the protocol overhead for a given task. One of the key reasons we see such interest in Web services is that Web services are well suited to enable a Service-Oriented Architecture (SOA). When using Web services to build an SOA, the solutions consist of collections of autonomous services, identified by URLs, with interfaces documented using WSDL, and processing well-defined XML messages. SOA is a natural complement to the object-oriented (OO), procedural and data centric approaches to solution implementation. Indeed, when creating an SOA system, the individual services are typically constructed using one or more of these technologies.

A Service-Oriented Architecture differs from OO and procedural systems in one key aspect: *binding*. Services interact based on *what* functions they provide and *how* they deliver them. OO and procedural systems link elements together based on type or name. The following sections provide more detail.

Services are described by schema and contract not type

Unlike previous systems, the Web service model does not operate on the notion of shared types that require common implementation. Rather, services interact based solely on contracts (WSDL/BPEL4WS for message processing behavior) and schemas (WSDL/XSD for message structure). This enables the service to describe the structure of messages it can send and/or receive and sequencing constraints on these messages. The separation between structure and behavior and the explicit, machine verifiable description of these characteristics simplifies integration in heterogeneous environments.

Furthermore, this information sufficiently characterizes the service interface so that application integration does not require a shared execution environment to create the messages structure or behavior.

The service-oriented model assumes a fully distributed environment where it is difficult, if not impossible, to propagate changes in schema and/or contract to all parties that have encountered a service. Service-orientation implies that contracts and schema should remain backward compatible and may contain information that is incompletely understood by particular processing systems.

For that reason, the contract and schema technologies designed for use in service-oriented designs enable more flexibility than traditional object-oriented interfaces. In particular, services use features such as XML element wildcards (e.g., xsd:any), schema extensions and optional SOAP header blocks to evolve services in ways that do not break deployed applications. These characteristics are the key to the composability of Web services.

Service compatibility is more than type compatibility

Procedural and Object-oriented designs typically equate type compatibility with semantic compatibility. Service-orientation provides a richer model for determining compatibility. Structural compatibility is based on contract (WSDL and optionally BPEL4WS) and schema (XSD) and can be validated. Moreover, the advent of WS-Policy provides for additional automated analysis of the service assurance compatibility between services. This is done based on explicit assertions of capabilities and requirements in the form of WS-Policy statements.

Using WS-Policy, services describe their service assurance capabilities and requirements in the form of a machine-readable policy expression containing combinations of assertions. This allows service to select each other based on "how" or "with what quality" they deliver their contracts.

Policy assertions are identified by stable and globally unique names whose meaning is consistent in time and space no matter which service the assertion is applied to. Policy assertions may also have parameters that qualify the exact interpretation of the assertion.

Service-orientation assumes that bad things can and will happen

Some previous approaches to distributed applications explicitly assumed a common type space, execution model, and procedure/object reference model. In essence, the "in-memory" programming model defined the distributed system model.

Service-orientation simply assumes that the services execute autonomously and there is no notion of local execution or common operating environment. For this reason, an SOA explicitly assumes that communication, availability, and type errors are common.

To maintain system integrity, service-oriented designs explicitly rely on a variety of technologies to deal with asynchrony and partial failure modes. Techniques such as asynchronous messaging, transactions, reliable messaging, and redundant deployment are the norm in a service-oriented systems.

Moreover, unlike the in-memory model, service-orientation assumes that not only that an incoming message may be malformed, but also that it may have been transmitted for malicious or completely unexpected purposes. Consequently service-oriented systems protect themselves by placing the burden of proof on *all* message senders by requiring applications to prove that the required rights have been granted to the sender. Consistent with the notion of service autonomy, service-oriented architectures typically rely on administratively managed trust relationships to avoid per-service authentication mechanisms common in classic web applications.

Service-orientation enables flexible binding of services

One of the core concepts of *service-oriented architecture* (SOA) is flexible binding of services. More traditional procedural, component and object models bind components together through references (pointers) or names. An SOA supports more dynamic discovery of service instances that provides the interface, semantics and service assurances that the requestor expects.

In procedural or object-oriented systems, a caller typically finds a server based on the types it exports or a shared name space. In an SOA system, callers can search registries such as UDDI for a service.

1. That is message compatible with the caller's requirements. Compatibility can occur through WSDL or matching messages from well-known XML Schemas.
2. That documents support for service assurances that the caller requires. For example, the caller may desire certain approaches to security or transactions.

The loose binding with respect to the implementation of the service that enables alternative implementations of behavior can be used to address a range of business requirements. For example, the alternative implementations might correspond to alternative vendors in a supply chain enabling more rapid response to changing market conditions. Similarly the alternative

implementation might be geographically distributed data centers enabling disaster tolerance.

Web Service Specifications and Functions

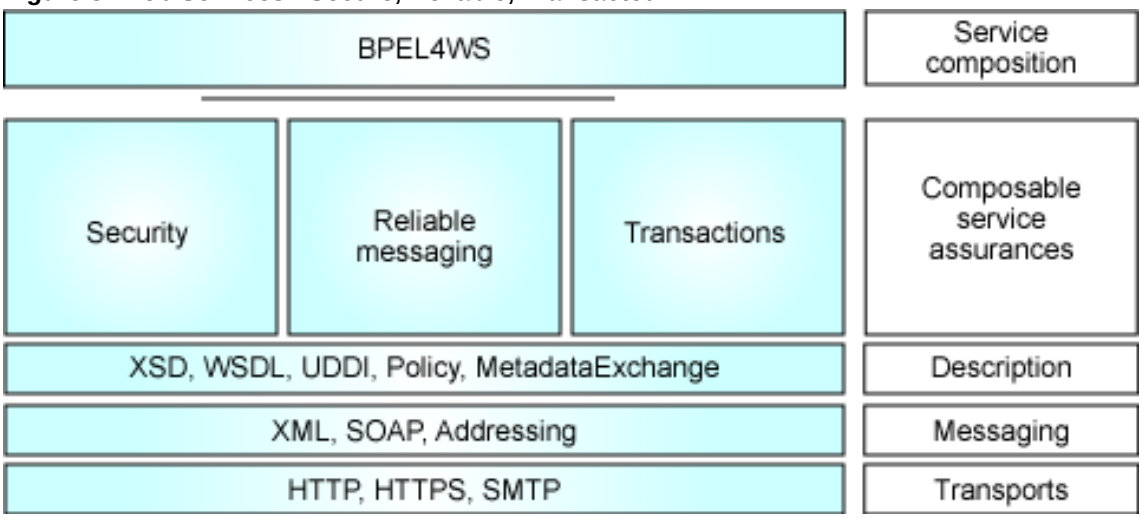
This section provides an overview of the Web service specifications.

A Composable Approach to Web Services

This section briefly describes the Web service specifications that are available. We explain their value to solution providers, their role in a broader architecture and how they compliment each other.

The following figure provides a high-level grouping of the Web service specifications published by IBM, Microsoft and others. Note that this figure is not meant to imply a strict layering between the groups; instead it is intended to provide an intuition about the relationships between functional areas. For example, message security does not require Description and similarly Description is a useful development time concept for Messaging.

Figure 3. Web Services - Secure, Reliable, Transacted



The Basics - Transports and Messaging

If I send you a letter written in French but you were expecting a telephone call in English, we will not communicate. Interoperability of Web services faces the same problem; we address this by providing a common set of transports and messaging technology.

Moreover, to ensure these technologies are effective in practice, IBM, Microsoft, and others created the Web Services Interoperability Organization (WS-I). Recently the WS-I released a basic profile that formally documents interoperable Web service transport and messaging mechanisms.

3.2.1 Transports - HTTP, HTTP/S, SMTP

This set of specifications defines the core communication mechanisms for moving raw data between Web services.

HTTP, HTTP/S, and Simple Mail Transport Protocol (SMTP) are examples in this group. Web service implementations may additionally support other transports, but it is critical to provide support for standard, interoperable protocols.

3.2.2 Message Formats - XSD

The next group of specifications defines interoperable mechanisms for encoding Web service messages for transport. The transports move blocks of "bytes" between services. This is only useful if the participants can convert the bytes into useful data structures that their application processes.

The messaging specification group defines how to format messages properly. XML and XML Schema definitions provide the

mechanism for abstractly agreeing on message (data) structures. SOAP defines the standard encoding for representing XML messages in the byte information that services exchange over transports.

3.2.3 WS-Addressing

Messages and responses both go somewhere and come from somewhere. WS-Addressing provides an interoperable, transport independent approach to identifying message senders and receivers. WS-Addressing also provides a finer grain approach to identifying specific elements within a service that send or should receive a message.

Today most systems using Web services encode the destination for a Web service message with a URL that is placed in the HTTP transport. The destination for the response is determined by the return transport address. This approach builds on the basic browser-server model of HTTP.

Using today's approach, the source and destination information are not part of the message itself. This can cause several problems. The information can be lost if a transport connection terminates (for example, if the response takes a long time and the connection times out) or if the message is forwarded by an intermediary such as a firewall.

WS-Addressing provides a mechanism to place the target, source and other important address information directly within the Web service message. In short, WS-Addressing decouples address information from any specific transport model.

In many scenarios messages are targeted directly to a service and the addressing information in the message can be described simply using a URL. But in practice, we often find that messages are targeted to specific elements or resources within a service. For example, a coordination service might be coordinating many different tasks. The coordinator needs to associate most incoming messages with a specific task instance that it manages and not the coordination service itself.

WS-Addressing provides a simple yet powerful mechanism called an *endpoint reference* for addressing entities managed by a service. While such information could be encoded in an ad-hoc manner within the URL of the service, the endpoint references provides a standard XML element that enables a structured approach to encoding this fine-grained addressing.

The combination of fine-grain control over addressing coupled with the transport-neutral encoding of the message source and destination enables Web service messages to be sent across a range of transports, through intermediaries, and it enables both asynchronous and extended duration communication patterns.

WS-Addressing also enables a sender to indicate where a response should go in a transport-independent manner. The response to a message may not necessarily go to the sender. In HTTP for example, without WS-Addressing it is impossible to specify that the response should be sent elsewhere.

These enhancements to the messaging model enable Web services to be used to support many business scenarios. For example, certain banking tasks require human review for approval at certain steps. There are usually many active instances of the task at any point in time. WS-Addressing provides a general mechanism to associate incoming or outgoing messages with specific tasks. The mechanism that the service uses is transparent to those using the service through an endpoint reference.

Description

The transport and message specifications allow Web services to communicate using messages. But how do the participants know what the messages are? How does a Web service document or describe the messages it sends and receives? Using a Web service requires an understanding of the messages the Web service will consume and produce - the *interface* for the Web service. The description group of specifications enables a Web service to express its interface and capabilities.

In addition to message interoperability; these specifications also enable *development tool interoperability*. The description specifications provide a standard model that allows different tools from different vendors to collaboratively support developers. In the same way that Web services isolate partners from implementation and infrastructure choices, the description specifications isolate partners from development tool choices.

3.3.1 WSDL

The Web Services Description Language (WSDL) and the XML Schema (XSD) are the base specifications in this group. XML

Schema allows developers and service providers to define XML types for data structures, e.g. a purchase order, and messages, e.g. the CreatePO message. WSDL allows a Web service to document the messages it receives and sends. In other words, what "actions" or "functions" the service performs in terms of the messages it receives and sends.

WSDL provides support for a range of message interaction patterns. It supports one-way input messages that have no response, request/response, and one way sends with or without a response. The last two patterns enable a service to specify other services that it needs.

Proposed WSDL enhancements provide support for documenting protocols and message formats a service supports, and the service's address.

3.3.2 WS-Policy

WSDL and XSD definitions often do not provide enough information to call a Web service. WSDL and XSD define the service's interface syntax but they do not express information about how the service delivers its interface or what the service expects of the caller. For example, does the service require security or implement transactions?

WS-Policy enables a service to specify what it expects of callers and how it implements its interface. WS-Policy is critical to achieve interoperability at the higher-level functional operation of the service. Security, transactions, reliable messaging and other specifications require concrete WS-Policy schema. These allow services to describe the functional assurance that they expect from and provide to callers.

The WS-Policy framework provides a base model for defining policy expressions.

WS-Policy supports a grammar for aggregating policy statements and allows the construction of more flexible and complete sets of policy.

WS-PolicyAttachment specifies how to associate a policy set with XML messages and WSDL elements (operations and portTypes).

Together WS-Policy and WS-PolicyAttachment provide the framework. Individual specifications define their domain specific policy statements and schema.

Finally, *WS-PolicyAssertions* provides a foundational set of common policy statements that can be used to achieve interoperability.

3.3.3 Obtaining Descriptions

XML, XSD, WSDL and WS-Policy support describing the interface and service assurances for a service. But, how do potential users of the service find this information?

Currently, the most common approach is through e-mail exchanges or word of mouth. A more general purpose, scalable model is necessary. There are two options, the service may go directly to the service to obtain information using *WS-MetadataExchange* or it may choose to use a *UDDI* service that aggregates this information for multiple target services.

Developers use WS-MetadataExchange when they have a reference to a service and need to understand what it does. Developers use UDDI when they want to find a reference to a service that supports a specific set of functions.

3.3.4 WS-MetadataExchange

As described above, services typically provide information such as WSDL, WS-Policy, and XSD, that describe the service itself. Collectivity we refer to information about the service as metadata. The WS-MetadataExchange specification enables a service to provide metadata to others through a Web services interface. Given only a reference to a Web service, a potential user can access a set of WSDL/SOAP operations to retrieve the metadata that describes the service. Clients can use WS-MetadataExchange at design time, when building their clients, or at runtime.

3.3.5 UDDI

Often it is useful to collect metadata about a collection of services and to make the information available in a form that is searchable. Such metadata aggregation services are a useful repository in which organizations can publish the services they provide, describe the interfaces to their services, and enable domain-specific taxonomies of services. The Universal Description and Discovery Interface (UDDI) specification defines a metadata aggregation service.

Solutions can query UDDI at design time to find services compatible with their requirements. The developers may use these services in the definition of their BPEL4WS workflows, for example. Solutions can also query UDDI at runtime. In this scenario, the caller "knows" the interface it requires and searches for a service that meets its functional requirements or is provided by a well-known partner.

Note that one of the mechanisms that might be used to populate a UDDI service with metadata is to acquire the metadata from services using WS-MetadataExchange.

Service Assurances

Web services have generated so much enthusiasm in part because of their ability to bridge disparate systems. Developers have produced many fully functional solutions using the base capabilities of transport, messaging and description. However, to be accepted by developers creating more powerful integration solutions, Web services must provide functionality to ensure the same level of *service assurances* provided by more traditional middleware solutions. It is not enough to simply exchange messages. Applications and services reside in middleware and systems that provide valuable higher-level functions such as security, reliability, and transacted operations. Web services must provide a mechanism for interoperability between these functions.

Security

This family of specifications is critical to cross-organization Web services. These specifications support authentication and message integrity, confidentiality, trust, and privacy. They also support federation of security between different organizations.

3.5.1 WS-Security

WS-Security is the basic building block for secure Web services. Today, most distributed Web services rely on transport level support for security functions. Examples are HTTP/S and BASIC-Auth authentication. These approaches to security provide the minimum necessary for secure communication. The level of function they provide, however, is significantly less than that provided by existing middleware and distributed environments.

Two examples highlight the deficiencies of BASIC-Auth and HTTP/S.

- A sends a message to service B. B partially processes the messages and forwards it to service C. HTTP/S allow authentication, confidentiality and integrity between A-B and B-C. However, C and A cannot authenticate each other, or hide information from B.
- For A, B and C to use BASIC-Auth for authentication. They must share the same replicated user and password information. This is unacceptable in many scenarios.

WS-Security solves these problems. It supports:

- Signed, encrypted security tokens. A can generate a token that C can verify as having come from A. B cannot forge the token.
- A can sign selected elements or the entire message. This allows B and C to confirm that the message has not changed since A sent it.
- A can seal the message or selected elements. This ensures that only the intended service for those elements can use the information. This prevents B from seeing information intended for C and vice-versa.

WS-Security uses existing security models (Kerberos, X509, etc). The specifications concretely define how to use the existing models in an interoperable way. Multi-hop, multi-party Web service computations cannot be secure without WS-Security.

3.5.2 WS-Trust

Security relies on pre-defined trust relationships. Kerberos works because participants "trust" the Kerberos Key Distribution Center. PKI works because participants trust the root certificate authorities. WS-Trust defines an extensible model for setting up and verifying trust relationships.

The key concept in WS-Trust is a *Security Token Service (STS)*. An STS is a distinguished Web service that issues, exchanges and validates security tokens. WS-Trust allows Web services to set up and agree on which security servers they "trust," and to rely on these servers.

The STS has broad applicability in that it can be used to issue security tokens that make a wide range of assertions. In many cases it will be used to issue the same assertions but in different formats. For example, an STS might issue a Kerberos token asserting that the key holder is Susan and it might do this based on an X.509 certificate issued by a trusted Certificate Authority. This enables organizations using different security technologies to federate. An STS might also issue a security token asserting that the key holder is a member of the group BankTellers based on an incoming security token that asserts an identity claim.

3.5.3 WS-SecureConversation

Some Web service scenarios only involve the short sporadic exchange of a few messages. WS-Security readily supports this model. Other scenarios involve long duration, multi-message conversations between the Web services. WS-Security also supports this model, but the solution is not optimal.

There are two sub-optimal usages of WS-Security in these scenarios:

- Repeated use of computationally expensive cryptographic operations such as public key validation.
- Sending and receiving many messages using the same cryptographic keys, providing more information that allows brute force attacks to "break the code."

For these reasons, protocols like HTTP/S use public keys to perform a simple negotiation that defines *conversation specific keys*. This key exchange allows more efficient security implementations and also decreases the amount of information encrypted with a specific set of keys.

WS-SecureConversation provides similar support for WS-Security. Participants often use WS-Security with public keys to start a "conversation" or "session," and use WS-SecureConversation to agree on session specific keys for signing and encrypting information.

3.5.4 WS-Federation

WS-Federation allows a set of organizations to establish a single, virtual security domain. For example, a travel agent, an airline and a hotel chain may set up such a federation. An end-user that "logs into" any member of the federation has effectively logged into all of the members. WS-Federation defines several models for providing federated security through protocols between WS-Trust and WS-SecureConversation topologies.

Additionally, customers often have "properties" when they deal with an enterprise. An example is a preference for window or aisle seats, or a midsize car. WS-Federation allows the members to set up a federated property space. This allows each participant to have secure controlled access to each member's property information about the end-users.

Properties and information about individuals may be closely held for privacy protection or because the information provides a competitive advantage to a specific member. To support these requirements, WS-Federation supports a *pseudonym model*. Users that have authenticated to the travel agency have agency generated "aliases" in their interactions with the airline or hotel. This protects the privacy of the end-user and the competitive advantage that the travel agency may gain by knowing user properties.

Reliable Messaging

In an Internet world, almost all communication channels are unreliable. Messages disappear. Connections break.

Without a reliable messaging standard, Web service application developers must build these functions into their applications. The basic approaches and techniques are well understood, for example many operating and middleware systems ensure messages have unique identifiers, provide sequence numbers, and use retransmission when messages are lost. If application Web service developers implement these models in their applications. They may make different assumptions or design choices, resulting in little if any reliable messaging.

3.6.1 WS-ReliableMessaging

WS-ReliableMessaging defines mechanisms that enable Web services to ensure delivery of messages over unreliable communication networks.

WS-ReliableMessaging ensures services implement interoperable approaches, and also enables runtime vendors to ease application development by providing services that implement the protocols. This significantly simplifies the task of application development. Business logic then has far fewer error conditions that it must handle.

Finally, the industry has a rich set of message-oriented middleware for reliably routing and distributing messages. Each implementation uses proprietary protocols. WS-Reliable Messaging protocols allow different operating and middleware systems to reliably exchange message. Thus, it supports bridging two different infrastructures into a single, logically complete, end-to-end model.

Transactions

A complex business scenario may require multiple parties to exchange multiple sets of messages. An example is a set of financial institutions setting up a financial offering that involves insurance policies, annuities, checking accounts and brokerage accounts. The multiple messages exchanged between participants constitute a logical "task" or "objective."

For success, the parties must be able to:

1. Start new coordinated tasks.
2. Associate operations with their logical task. The parties may be setting up multiple accounts for different customers at the same time.
3. Agree on the outcome of the computation. For example, does everyone agree that the financial packages were set up?

WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity support these requirements.

3.7.1 WS-Coordination

WS-Coordination is a general mechanism for starting and agreeing on the outcome of multiparty, multi-message Web service tasks. WS-Coordination has three key elements:

1. A message element called a coordination context that flows on all messages that Web services exchanges during the computation. The coordination context contains the WS-Addressing endpoint reference to the coordination service and it in turn contains information to identify the specific task being coordinated.
2. The *coordinator service*. The coordinator service provides a service, described using WSDL, that provide the ability to start a coordinated task, terminate a coordinated task, allow a participant to register in a task, and produce a coordination context that is part of all messages within a group.
3. The coordination service also includes an interface, defined in WSDL, that participating services use in order to be informed of the outcome of the coordinated task.

A Web service that receives a message with a new coordination context registers with the coordinator service in the context in order to receive outcome information. Other specifications may augment this framework for domain and assurance specific requirements.

WS-Coordination is a general framework and capability. WS-AtomicTransaction and WS-BusinessActivity extend this framework to allow the participants in the distributed computation to robustly determine outcomes.

3.7.2 WS-AtomicTransaction

WS-AtomicTransaction defines a specific set of protocols that plug into the WS-Coordination model to implement traditional two-phase atomic transaction protocols. It is important to note that the atomic, two-phase model is only with respect to the services involved. Sites or infrastructure offering services may advertise two-phase commit, but use some other intra-enterprise model like compensation or versioning. This freedom makes a simple two-phase commit model more useful for long-running Internet computations.

3.7.3 WS-BusinessActivity

WS-BusinessActivity defines a specific set of protocols that plug into the WS-Coordination model to implement long-running, compensation-based transaction protocols. While BPEL4WS defines a transaction model for business processes, it is WS-BusinessActivity that specifies the corresponding protocol rendering. This, again, is an example for the composability of the Web services specifications.

Service Composition

The uppermost element in the Web service layering is *service composition*. Service composition allows developers to "compose" services that exchange SOAP messages and define their interface in WSDL and WS-Policy into an aggregate solution. The aggregate is a composed Web service.

3.8.1 BPEL4WS

The Business Process Execution Language for Web Services (BPEL4WS) specification supports service composition. It enables developers to define the structure and behavior of a set of Web services that jointly implement a shared business solution. Each element of the set of services defines its interface using WSDL and WS-Policy. The composed solution is itself a Web service, which supports HTTP/SOAP messages and defines its interface using WSDL and WS-Policy.

Composition has three aspects: *structure*, *information* and *behavior*. BPEL4WS introduces three constructs supporting each composition aspect.

A *partnerLink* defines a named association between the composite service and a Web service that participates in the overall solution. The composite service and participating service define their interfaces to each other using WSDL and WS-Policy. An example might be an association between a manufacturing company and a supplier.

The partnerLink concept and the WSDL/WS-Policy interfaces between the composition and partners define the *structure* of the service composition. They define the types of services that collaborate to form the composition, and which messages they exchange with which levels of assurance (security, transactions, etc.)

BPEL4WS also provides support for defining the *information* of the service composition. BPEL4WS defines the concept of a container. The composite service defines a set of containers, each of which has an XSD definition. The current state of a specific service is the state of its containers. This defines what messages it has received or sent.

Finally, BPEL4WS supports defining the behavior of the composite service by the concept of an *activity*. A BPEL4WS defined service is a set of activities or "steps," which define the behavior of the service. The most basic activities are sending a message to a partner or receiving a message from a partner. Each message corresponds to a container. BPEL4WS provides support for moving data between containers.

One key aspect of BPEL4WS activities is that BPEL4WS provides special support for defining externally visible (public) behavior of services by allowing controlled use of non-deterministic behavior. For instance the fact that a credit check is performed in a specific way in the decision process for accepting a PO may be a private matter for a supplier. BPEL4WS allows the decision process to be hidden by dropping the credit check behavior from the process description but showing that the response to the PO may be either acceptance or rejection. This type of *abstract* process can be used in conjunction with WSDL to define interoperable business protocols between business partners and for vertical industry domains such as supply

chain.

BPEL4WS also supports several approaches to controlling the flow of execution of activities. These include sequencing and graph based flows. BPEL4WS support predicates on containers to determine which control paths the composite service follows.

To summarize, BPEL4WS makes two additions to the previously defined Web service specifications.

1. BPEL4WS extends WSDL and WS-Policy support for describing services. BPEL4WS support combining Web services into aggregate services, and documenting the associations between services, such as the information flow and behavior. This provides support for interoperability between higher layer tools supporting collaborative design of Web services.
2. BPEL4WS is an *execution language*. BPEL4WS allows developers to fully specify the behavior of a composite Web service. IBM, Microsoft, and other partners will provide environments that execute BPEL4WS documents and support design and execution time binding to partners.

Web Services in Practice - An Example

The following scenario shows how the WS specifications can be used together to create Web services that solve real-world needs. The scenario provides an example of the powerful functionality available to developers because of the composability of the different WS specifications.

The Web services described in this scenario were created for a joint IBM-Microsoft demonstration of the technology held on September 17, 2003. They were used to create an application that is interoperable, secure, reliable, and transacted; and that spans organizational boundaries.

The demonstration shows a running example of a federated order processing and Vendor Managed Inventory (VMI) system for a car dealer ordering a part from an automobile manufacturer; the manufacturer in turn obtains parts from a supplier operating multiple warehouses. All application to application communications in the system were built exclusively using the Web service protocols described previously and running on a collection of computers with IBM and Microsoft software.

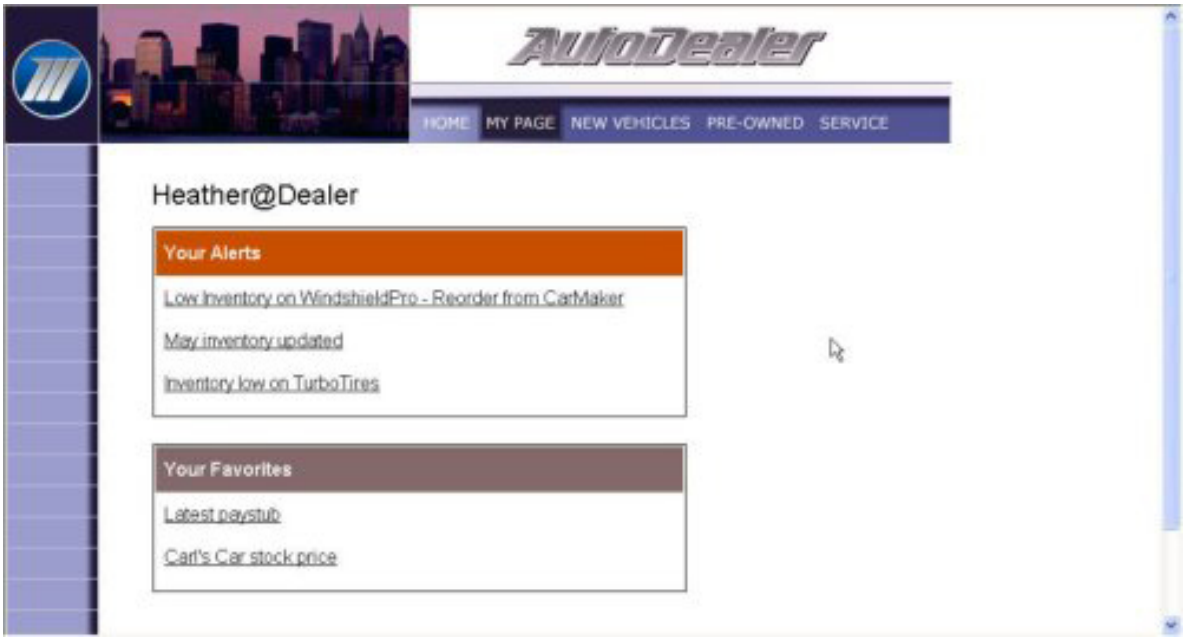
The scenario deals with some of the most common aspects of conducting business - the interaction between a retail business, its wholesaler, and the wholesaler's supplier. The scenario shows how different WS specifications can be composed to automate business process essentials such as:

1. Authentication to enforce security (WS-Security)
2. Federation of trust between different organizations (WS-Trust and WS-Federation)
3. Exchange of data to complete a transaction (WS-AtomicTransaction)
4. Assurance that orders have been submitted through reliable messaging (WS-ReliableMessaging)

Part 1: The Customer Experience

The example begins with Heather, an employee at a company called Auto Dealer, logging onto her dealership's secure Intranet web site. This web site is built using standard, off-the-shelf web technologies. Heather enters the site using her browser. Access to the site is password protected.

Figure 4. Heather logs onto her company's secure intranet Web site, and navigates to her customized My Page.



Heather clicks on My Page. In the background the application gathers information from the Auto Dealer's inventory database. If inventory levels for an item fall below a defined threshold, a report is generated and listed in the "Your Alerts" display of Heather's page.

Heather sees that her company has a low inventory on WindshieldPro wiper blades.

Heather clicks on the link and is seamlessly redirected to a secure Web page on the Auto Manufacturer extranet, where Heather can place an order. The experience is seamless because the Auto Dealer software is based on Web services. The Web service linking the Auto Dealer's intranet with the Auto Manufacturer extranet was composed using WS-Federation. WS-Federation ensures that security credentials granted by one site is honored by a second site.

Here's how this works. The Auto Dealer and Auto Manufacturer have agreed to federate their sites. WS-Federation coordinates a series of server-to-server communications. As soon as Heather clicked on the WindshieldPro link to take her to the Auto Manufacturer's Web page, The Auto Manufacturer's Web page server queried its authorization service, which in turn queried Auto Dealer's authorization service. The Auto Dealer authorization service confirms Heather is an authorized user, transmitting credentials, along with the name of Heather's dealership, back to Auto Manufacturer's authorization service, which grants Heather access. This happens so seamlessly, that all Heather notes is that she has gone from one Web page to another.

The Web service also queries the Auto Manufacturer customer database for ordering information linked to Heather's account. The information is presented in a personalized "My Page" Auto Manufacturer Web page.

Figure 5: Composing a Web service using WS-Federated allows Heather to seamlessly move from her personalized page at Auto Dealer to her personalized page at Auto Manufacturer.

Heather's personalized Web page on the Auto Manufacturer extranet allows her to see that she currently has no outstanding orders; she has one order (for 50 SuperTires) in transit; and that her list of completed orders includes 20 units of CDPlus and 50 units of Leather Cleaner.

Heather clicks on "Create New Order", and a new page opens, prepopulated with the part she wants - WindshieldPro wiper blades, and the ordering date. All she needs to enter is the quantity. All other information needed to complete the order is populated from the Auto Manufacturer database.

Figure 6. An order is easily placed because much of the ordering data has already been imported from Heather's file on the Auto Manufacturer customer database.

Heather submits the order and either searches for additional items to purchase, or clicks Logout, to end her session and prevent anyone else from placing an order from her unattended computer.

Note that composing the Web service with WS-Federation provided both Auto Dealer and Auto Manufacturer with lower administrative costs and end-to-end security. Without this technology, Auto Manufacturer would have had to maintain a list of all authorized dealership employees and their passwords. This would be costly, prone to errors, and reduce the security of the application.

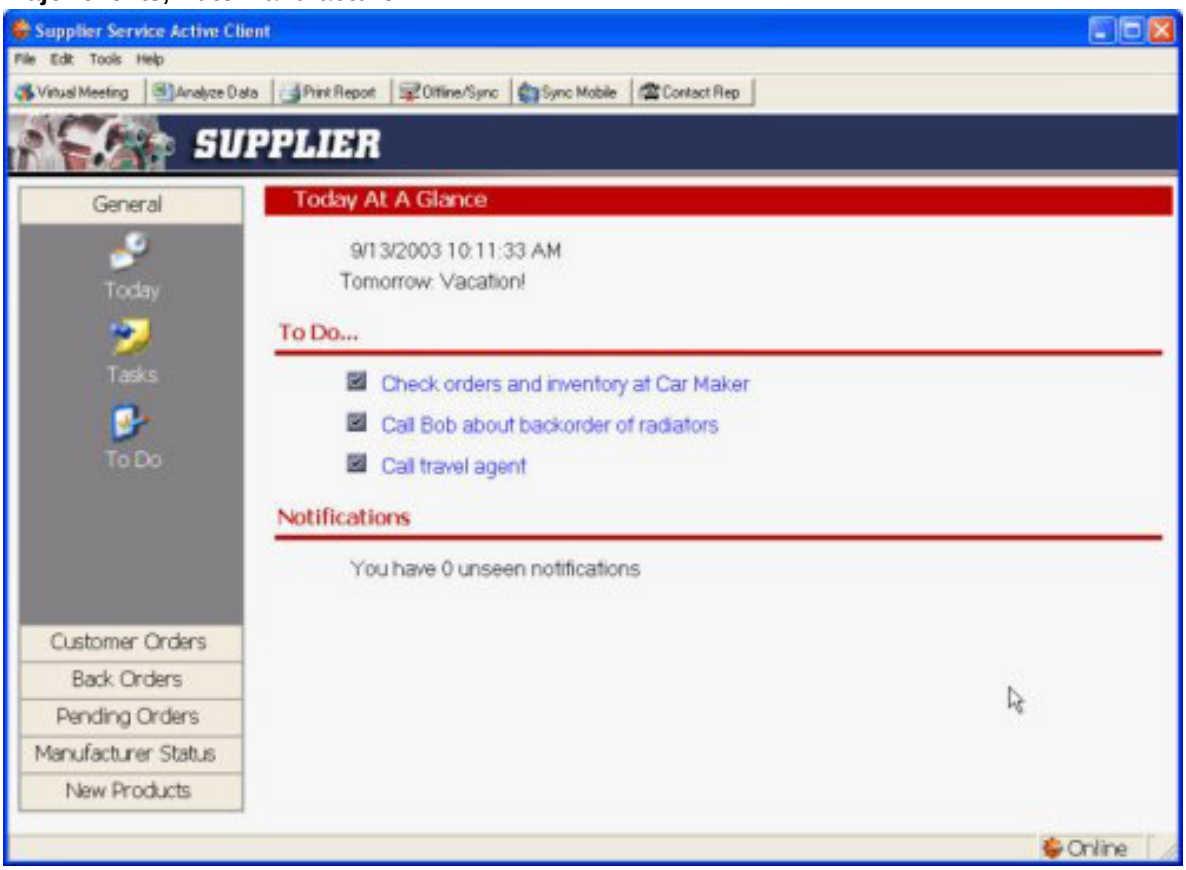
For instance, if Heather quit her job, her user account would be removed at Auto Dealer. But, if the administrator at the dealership forgot to contact Auto Manufacturer about her departure, she would continue to have access to the purchasing site. With WS-Federation, this isn't an issue, because the only system that has to be changed is the Auto Dealer's Identity Provider service. The Auto Manufacturer's systems, both the application and the Authorization Service, have no innate knowledge of Heather, her username, or her password.

Part 2: The Supplier Experience

Although Heather orders WindshieldPro wiper blades from Auto Manufacturer, it has been half a century since the company actually made its own blades. The WindshieldPro wiper blades come from a vendor, Supplier.

Tony is the sales representative for Supplier, and he begins his day by logging onto the Supplier's intranet, just as Heather logged onto the Auto Dealer intranet. But instead of using a standard web browser, Tony works with a Windows application that has built-in support for Web services.

Figure 7. Tony's personal page on the Supplier's intranet reminds him to check orders and inventory at one of his major clients, Auto Manufacturer.

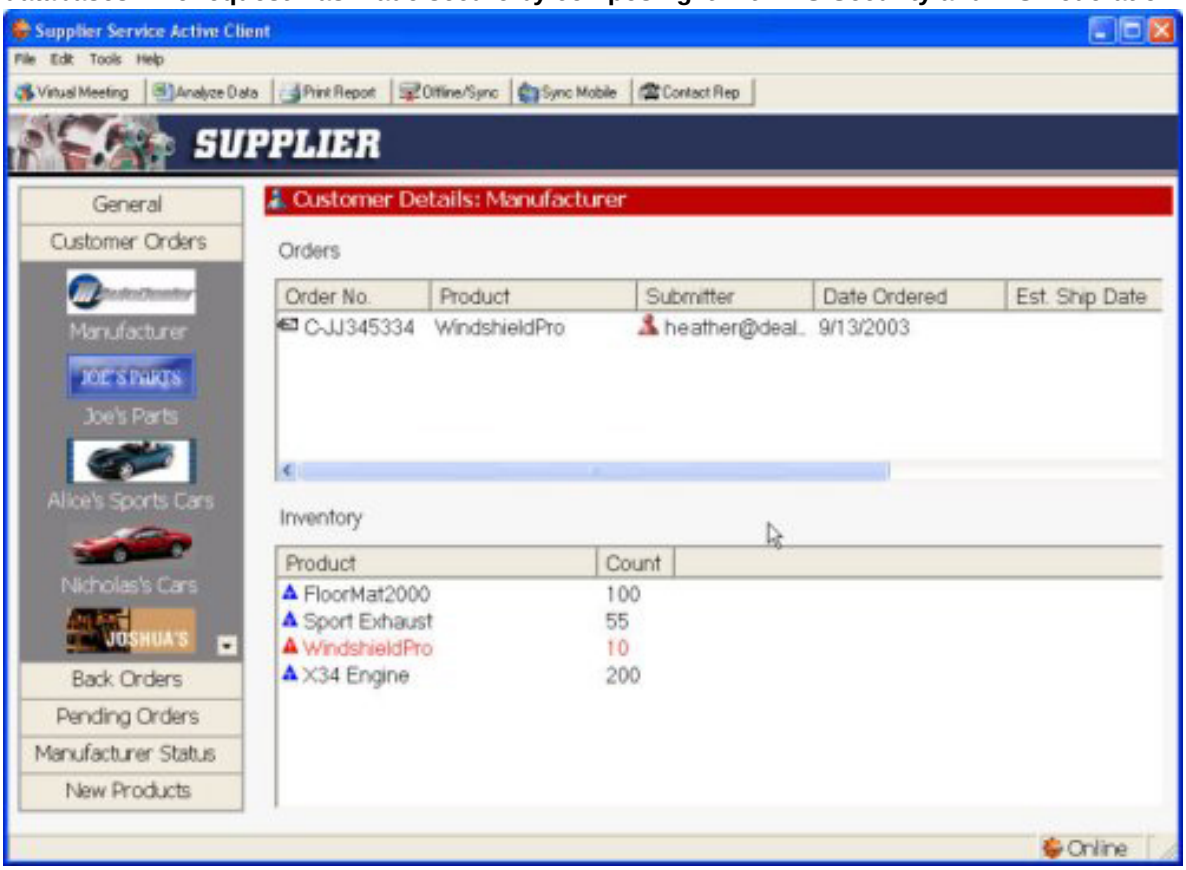


When Tony clicks to check orders and inventory, his application uses a Web service to ask for inventory data that Tony is allowed to access.

One key aspect of this application level Web services request for data is that it is composed with WS-Federation which authenticates his access to the Auto Manufacturer's extranet.

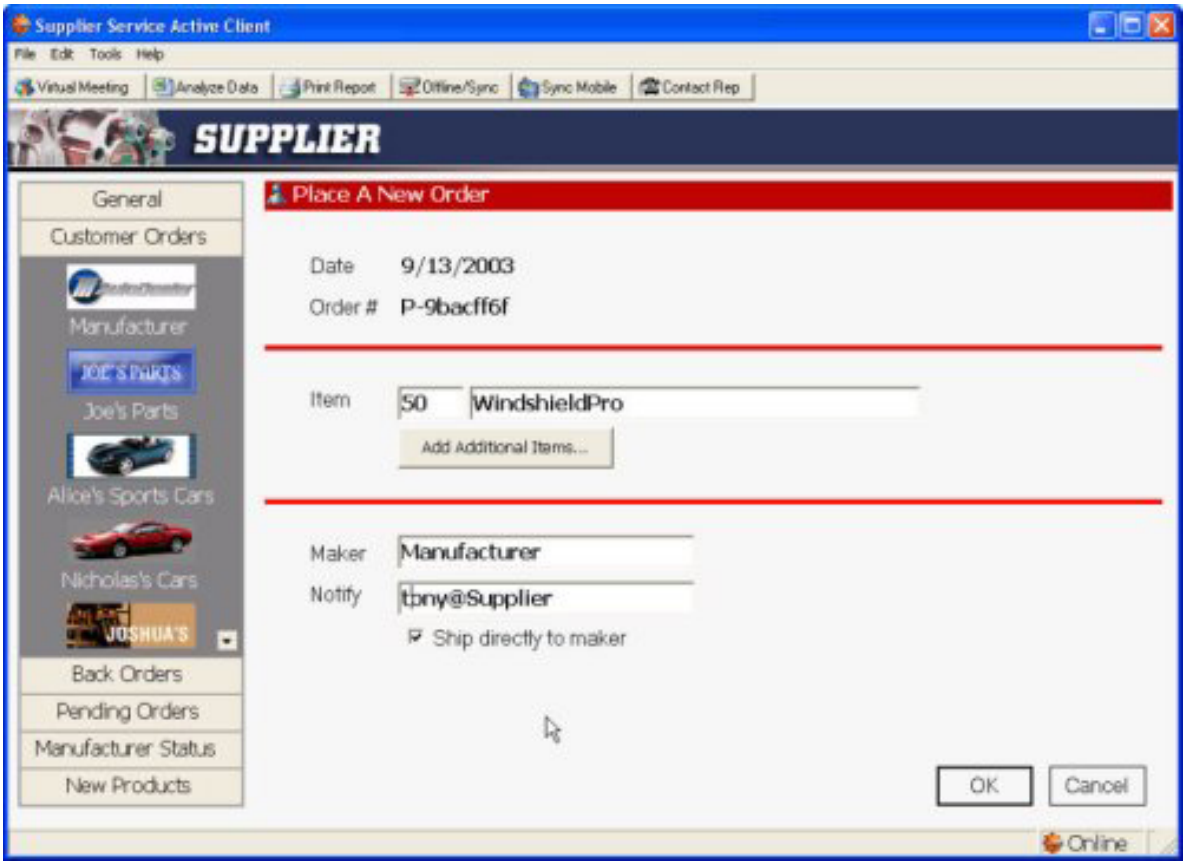
The Web service returns the results back to the Supplier's page where it is displayed by product type and current inventory count.

Figure 8. A Web service populates Tony's Supplier page with inventory data from Auto Manufacturer's inventory databases. The request was made secure by composing it with WS-Security and WS-Federation.



Supplier has entered into a just-in-time purchasing agreement with Auto Manufacturer. Tony is authorized to provide an automatic re-supply order as part of Vendor Managed Inventory (VMI) on Auto Manufacturer's behalf once inventory falls below 20. Tony clicks on WindshieldPro to place an order. All messages between Tony and Auto Manufacturer are protected because the application is supported by Web services composed with the protections of WS-Security.

Figure 9. A just-in-time agreement with Auto Manufacturer allows Tony to enter an order on the company's behalf.



Tony's application provides him with a screen to create requests to Supplier with a Auto Manufacturer purchase order. Tony orders 50 WindishieldPros wipers to be shipped directly to the Auto Manufacturer.

When Tony clicks OK, the Warehouse Service, a Web service composed with WS-AtomicTransactions, WS-Security, and WS-ReliableMessaging, attempts to place the order with another Web service, the subordinate Warehouse Services. When a response isn't immediately returned from the Warehouse Service (because of a temporary network outage) WS-ReliableMessaging continues to resend the query, until receiving a response.

When the warehouse service receives the order it relays them internally to the company's two physical warehouses. Since this involves databases between both warehouses, WS-AtomicTransaction is used to ensure transactional behavior between the databases.

The Warehouse application divides the orders among the subordinate Warehouse services to ensure inventory coverage, 70% of order goes to Warehouse 1 and the remaining 30% goes to Warehouse 2. The Root Coordinator in the Main Warehouse sends a message to the Root Coordinator of Warehouse 2 asking for 30% of the order. The Root Coordinator of Warehouse 2 checks it's inventory and sends a message to the Root Coordinator of the Main Warehouse that there is not enough inventory in stock.

The main Root Coordinator knowing that there is not enough inventory cancels the entire transaction and sends a message to Tony's Web service application indicating the status, inventory levels, and that the transaction was cancelled. Root coordinator starts to back out transaction and when complete goes back to the Warehouse to say the transaction has been canceled.

Tony, with the current inventory knowledge, sends another request to the Warehouse. The Root coordinator coordinates among other transactional entities (controller of other transactions) and submits this request to the 2 Warehouses going through the same process as before. We are using WS-Security to sign the entire message body so no matter which domain you are in you know that you are secure.

Now the Root coordinator commits the transactions, locks resources and completes the transaction. A message is sent back to Tony indicating that the transaction has completed successfully.

WS-AtomicTransaction composes with WS-ReliableMessaging and WS-Security in these messages, to offer a complete enterprise-ready distributed system.

Conclusions

IBM, Microsoft, and our partners are developing Web service specifications that can be used as the building blocks for a new generation of powerful, secure, reliable, transacted Web services.

These specifications are designed in a modular and composable fashion such that developers can utilize just the capabilities they require. This "component-like" composability will allow developers to create powerful Web services in a simple and flexible manner, while only introducing just the level of complexity dictated by the specific application.

These Web service technologies enable organizations to easily create applications using a Service-Oriented Architecture (SOA). Furthermore, IBM and Microsoft have demonstrated secure, reliable, transacted SOA applications that illustrate the richness of the business processes that can be created using this approach. Moreover, these demonstrations have been operating in a federated security environment on a heterogeneous collection of systems running IBM WebSphere and Microsoft .NET software.

We anticipate that these Web Service technologies will be available in operating systems and middleware, with tools that will make it even easier for developers to use these technologies. It will be exciting to see the applications that emerge as developers and organizations use these systems to create the next generation of Web services-based solutions.

Acknowledgements

We wish to acknowledge the following individuals that contributed to these ideas: (alphabetical) Tony Andrews, Bob Atkinson, Keith Ballinger, Don Box, John Brezak, Allen Brown, Felipe Cabrera, Erik Christensen, George Copeland, Michael Coulson, Giovanni Della-Libera, Brendan Dixon, Mike Dusche, Colleen Evans, Max Feingold, Jeff Frey, Henrik Frystyk Nielsen, Praerit Garg, Omri Gazitt, Scot Gellock, Josh Gray, Martin Gudgin, MaryAnn Hondo, Destry Hood, Efim Hudis, Tomasz Janczuk, Jim Johnson, Ryan Johnson, Gopal Kakivaya, Chris Kaler, Johannes Klein, Scott Konersmann, Brian LaMacchia, Dave Langworthy, Andrew Layman, Paul Leach, Al Lee, Frank Leymann, Rodney Limprecht, Joe Long, Steve Lucco, John Manferdelli, Ashok Malhotra, Jonathan Marsh, Steve Millet, Angela Mills, Tony Nadalin, Martin Nally, Karla Norsworthy, Stefan Pharies, Scott Robinson, Yordan Rouskov, Sujay Sahni, Jeff Schlimmer, Oliver Sharp, Yasser Shohoud, Dan Simon, Jeff Spelman, Keith Stobie, Satish Thatte, Robert Wahbe, Elliot Waingold, Richard Ward, Sanjiva Weerawarana, Hervey Wilson, Eric Zinda.