UNIT V

Metadata Management

Metadata is data about data, or more precisely, data about a software entity associated with it in some way. A Web service may have a variety of metadata associated with it, including data types and structures for messages, message exchange patterns for exchanging messages, the network addresses of the endpoints that exchange the messages, and any requirements for extended features such as security, reliability, or transactions.

Web services metadata is an important part of basic and SOA-based Web services solutions. In general, the more complex the application of Web services, the greater the need for metadata and comprehensive metadata management solutions.

Web services metadata and metadata management technologies include the following:

UDDI—. A registry and repository for storing and retrieving Web services metadata.

XML Schema—. For defining data types and structures.

WSDL—. For defining messages, message exchange patterns, interfaces, and endpoints.

WS-Policy—. For declaring assertions for various qualities of service requirements, such as reliability, security, and transactions.

WS-Addressing—. For defining Web service endpoint references and associated message properties.

WS-MetadataExchange—. For dynamically accessing XML, WSDL, and WS-Policy metadata when required.

These different kinds of metadata work together to define the characteristics of any Web service, from simple to complex. The metadata items are contained in XML files of varying definition and are typically stored in a directory, such as UDDI or LDAP, or in plain files for easy retrieval. All of the metadata items benefit from the use of consistent design and naming conventions, especially for enterprise solutions.

To organize Web services metadata, you generally start with the WSDL file and add information to or extend it. The WSDL defines the characteristics of the interaction between the service requester and provider.

Before a Web service is invoked, the requester needs access to sufficient metadata to generate a SOAP message, including optional and required headers, to meet the description of the service that the provider publishes. If the provider's service is a basic Web service, sufficient metadata simply includes the message data format, message exchange pattern, and provider's network address. If the provider's service uses extended features, the requester may need to locate and obtain additional metadata, such as policy schemas for security, reliability, or transactions, and endpoint references with their optional associated properties.

The fundamental metadata required to execute a Web service typically includes the WSDL file and the endpoint address at which the SOAP listener is available to receive (and optionally return) messages. The basic requirement is to publish the metadata for a Web service so that the service requester can find and use it. Initially, UDDI was proposed to meet this requirement, but UDDI did not succeed in realizing its original vision.

UDDI was designed to support dynamic discovery, in which the lookup of Web services metadata was a seamless part of a Web service execution. The original vision of UDDI was that a Web service requester could supply some general information, such as a business name, business classification, or business location, and receive a valid Web service description with which to execute the request. However, for various reasons including the difficulty of creating

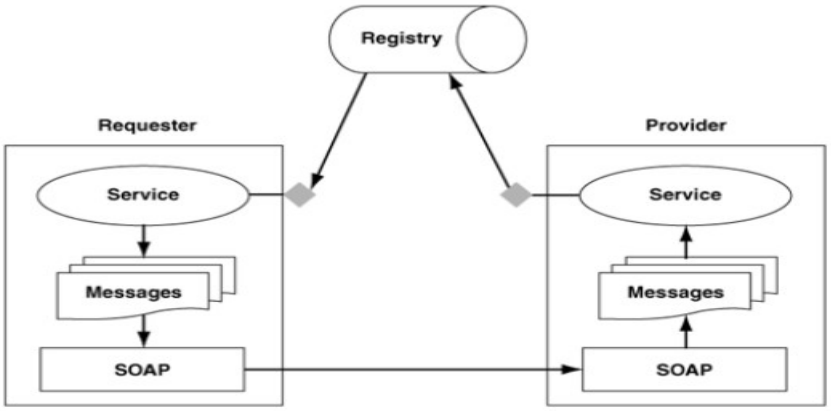
meaningful categorization information for the Web, UDDI did not achieve its goal of becoming the registry for all Web services metadata and did not become useful in a majority of Web services interactions over the Web.
Many other proposals were presented for Web services metadata management, including DISCO, Microsoft's original discovery specification, and WS-Inspection, a later effort by IBM. More recently, Microsoft and IBM have published WS-MetadataExchange as an alternative mechanism for Web services metadata discovery.
Any registry used for Web services metadata needs to support the general feature list of UDDI—that is, storage and retrieval of descriptions (or pointers to them) and associated metadata such as policies.

Metadata specifications:

XML
The main use of XML in Web services metadata is for defining data types and structures for SOAP messages
Ex:

```
<xs:schema targetNamespace="http://www.mybank.com/2005/05/schemas/AccountService.xsd"
    xmlns:xs=”http://www.w3.org/2001/XMLSchema”
 <xs:element name="CustomerInfo" type="tCustomerInfo"/>
     <xs:complexType name="tCustomerInfo">
          <xs:sequence>
               <xs:element name="AccountNumber" type="xs:double"/>
              <xs:element name="CustomerName" type="xs:string"/>
              <xs:element name="CustomerAddress" type="xs:string"/>
              <xs:element name="CustomerType" type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
<xs:element name="ConfirmationInfo" type="tConfirmationInfo"/>
      <xs:complexType name="tConfirmationInfo">
            <xs:sequence>
                 <xs:element name="AccountNumber" type="xs:double"/>
                 <xs:element name="ConfirmationCode" type="xs:string"/>
                 <xs:element name="ConfirmationType" type="xs:string"/>
                 </xs:sequence>
             </xs:sequence>
       </xs:complexType>
```

This example illustrates data types and structures for a customer information message, a confirmation message that can be used in conjunction with a composite Web service application to open a bank account.
As illustrated in Figure, a registry can be used as an intermediate storage mechanism between the requester and the provider. The provider can publish the description (and associated policy and data schemas) to the registry, and the requester can search the registry for the metadata it requires.



Registries are used for storing and retrieving service metadata.

Addressing

Two specifications define additional information for Web services references: WS-Addressing and WS-Message Delivery.

As shown in Figure, when a Web service interaction spans more than a single requester and provider (i.e., involving multiple providers), an addressing mechanism is required so that the initial provider can forward the request to the next provider, and so on. When the final provider (sometimes called the ultimate receiver) is reached, addressing information in the SOAP headers is used to return the reply to the initial requester.

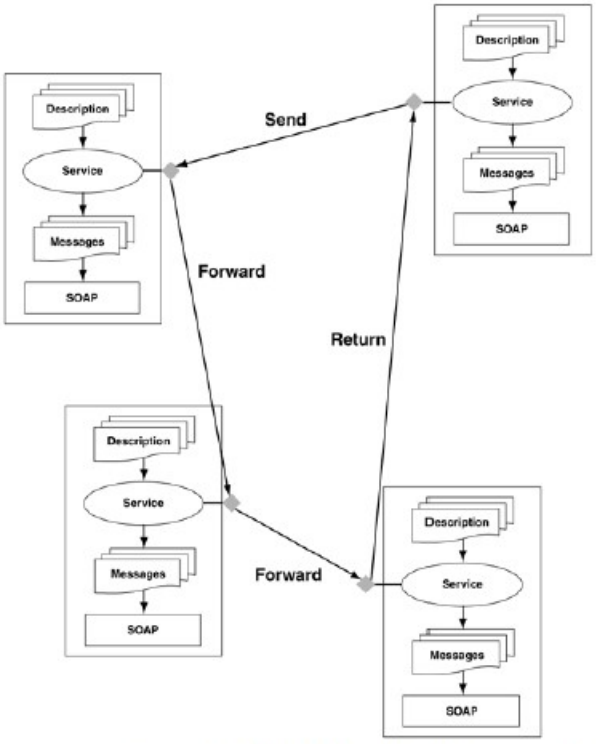


Figure 7-3. Addressing information allows complex exchange patterns.

A formal address also helps a message get back to the requester if there's a network failure and the response is lost. Addressing is also a requirement for many other specifications, including WS-BPEL, WS-AtomicTransactions, WS-Notification, WS-Eventing, WS-CAF, and WS-MetadataExchange.

WS-Addressing defines two mechanisms for SOAP headers that are typically provided by transport protocols and messaging systems:

Endpoint references—. Where to find a service on the network and any associated properties that may be necessary to interact successfully with that endpoint.

Message information headers—. Identifies the sender and optional reply address (when different).

Defining these mechanisms at the SOAP header level provides a better abstraction of the SOAP message from the underlying transport and makes multi-transport implementations of Web services easier. For example:

```
<wsa:ReplyTo>
    <wsa:Address>
        http://www.iona.com/Artix/BankExample
    </wsa:Address>
 </wsa:ReplyTo>
<wsa:To>
        http://www.iona.com/Artix/TransferFunds
</wsa:To>
<wsa:Action>
          http://www.iona.com/Artix/Samples/Transfer
</wsa:Action>
```

This example illustrates the WS-Addressing namespace wsa and shows the address from which the message is being sent (http://www.iona.com/Artix/BankExample), along with the address to which the reply is to be sent (http://www.iona.com/Artix/TransferFunds) and the action to be taken (Transfer). The body of the message would include the amount to transfer. Additional addresses could be included to route the message reply anywhere on the network.

WS-Addressing headers define an address for a Web service endpoint and also for an individual message destination. The message information headers carry information about where the message is from, where it's going, and where the reply (if any) should be sent.

```
<Header>
    <wsa:ReplyTo>
        <wsa:Address>
            http://www.iona.com/artix/examples/BankTransfer
        </wsa:Address>
     </wsa:ReplyTo>
    <wsa:To>
         http://www.NattyBank.com/accounts/Inbound
    </wsa:To>
   <wsa:Action>
        http://www.iona.com/newcomer/transferMillions
   </wsa:Action>
</Header>
```

This example illustrates a message sent from www.iona.com's bank transfer application to the Natty Bank's accounts department. The requested action is to transfer millions of dollars into Newcomer's bank account. The Action URI provides a way to express a semantic operation to be performed upon the message.
Addressing metadata is used to identify the target of a SOAP message, in particular, the:

- TCP address of the computer hosting the service (derived from the domain portion of the URL, which is what the domain part of iona.com or other Internet address translates to).
- Location on the computer of the service to be executed (often found by registering a Web service with a Web server, for example).
- Specific operation within the service to handle the message—receiving the message, processing the input data—and optionally returning a reply.

WS-MESSAGEDELIVERY

WS-MessageDelivery defines delivery properties, message identification, and message referencing for Web services message exchange patterns such as callback and broadcast. The message delivery properties are designed to work with WSDL-defined and other message exchange patterns. The WS-Message Delivery specification defines a Web service reference, abstract message delivery properties, a SOAP binding, and its relationship to WSDL.

```
<wsmd:MessageOriginator>
     <wsmd:uri>http://www.iona.com/Artix/BankExample </wsmd:uri>
</wsmd:MessageOriginator>
<wsmd:MessageDestination
       wsmd:wsdlLocation="http://www.NattyBank.com/wsdl
      <wsdlll:service name="myservice" wsmd:portType="myns:myPortType">
           <wsdlll:port name="myport" binding="myns:myBinding">
                <soapbind:address location="http://example.com/wsdl/impl"/>
           </wsdl11:port>
       </wsdl11:service>
</wsmd:MessageDestination>
<wsmd:ReplyDestination>
      <wsmd:uri>http://www.iona.com/Artix/BankExample/reply</wsmd:uri>
</wsmd:ReplyDestination>
<wsmd:FaultDestination>
       <wsmd:uri>http://www.iona.com/Artix/BankExample/fault</wsmd:uri>
</wsmd:FaultDestination>
<wsmd:MessageID>
        uuid:58f202ac-22cf-11d1-b12d-002035b29092
</wsmd:MessageID>
<wsmd:OperationName>TransferMillions</wsmd:OperationName>
```

This example illustrates the major parts of WS-MessageDelivery—the message originator and destination, the reply destination, a fault destination, and an operation name. These are all comparable to the major parts of WS-Addressing. However, WS-MessageDelivery adds a message ID in the form of a UUID.

Policy

A policy is an assertion about a service that describes one or more characteristics that a provider instructs a requester to follow. Policies can be expressed in a variety of ways, from simple statements of fact such as "strong authentication is required using Kerberos" to a set of rules evaluated in priority order that determine whether or not a requester can interact with a provider.

Policies are also intended as a vehicle to express service level agreements (SLAs), which are important in an SOA because they describe requirements for load balancing, availability, service delivery guarantees, and response time.

Policy assertions inform the requester about any additional information beyond "plain" WSDL (such as requirements for extended features) that may be needed to successfully invoke the provider's service. Additional information can include alternate transports, security requirements, whether or not a transaction is needed or accepted, and whether reliable messaging is required. The provider's service must publish the policy assertion information so that the requester can access it. A policy assertion provides the metadata in a way the service requester can understand and consume, similarly to the way in which the requester understands and consumes a provider's WSDL.

When a requester obtains the set of policy assertions from the provider (or from a policy repository), the effective policy (i.e., the one the requester will actually use) is calculated using the assertion information and a set of rules defining how the assertions are to be evaluated. Some policies might have higher priority than others, some are in effect only when others are in effect, and some might not be necessary at all under certain conditions.

Three mechanisms for expressing policy exist in various Web services specifications:

WS-Policy Framework—. Developed by BEA, IBM, Microsoft, and SAP.

Web Services Policy Language—. Created as a subgroup within the XACML Technical Committee at OASIS.

WSDL 2.0 features and properties—. Created by the WSDL Working Group at W3C.

WS-Policy refers to a set of three specifications:

WS-PolicyFramework. (often called WS-Policy for short because it's the "enclosing" specification for the other two)—Defines the overall model and syntax that can be extended by other specifications (such as WS-SecurityPolicy).

WS-PolicyAssertions—. Defines a basic set of assertions for policies, including whether an assertion is required or not.

WS-PolicyAttachment—. Defines how to attach policy assertions to WSDL files.

The WS-PolicyFramework specification defines a policy expression as an XML serialization consisting of three components:

- A top-level container element.
- Policy operators that group assertions.
- Attributes that determine policy usage.

The top-level container element is:

```
<wsp:Policy xmlns:wsp="..." xmlns:wsse="...">
. . .
</wsp:Policy>
```

The top-level element encloses the policy assertions and identifies the namespaces being used. In this example, there are two namespaces: one for WS-Policy and the other for WS-Security.
The following is an example of policy operators:

```
<wsp:ExactlyOne>
<wsp:All wsp:Preference="100">
. . .
</wsp:All>
<wsp:All wsp:Preference="1">
. . .
</wsp:All>
</wsp:ExactlyOne>
```

Policy operators enclose assertions, defining priority using the preference attribute and how many assertions from the list are to be enforced. Options include exactly one, all, or one or more.

The following is an example of attributes that determine policy usage for WS-Security:

```
<wsse:SecurityToken>
    <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
</wsse:SecurityToken>
<wsse:Integrity>
   <wsse:Algorithm Type="wsse:AlgSignature"
        URI=http://www.w3.org/2000/09/xmlenc#aes />
 </wsse:Integrity>
<wsse:SecurityToken>
    <wsse:TokenType>wsse:x509v3</wsse:TokenType>
</wsse:SecurityToken>
<wsse:Integrity>
    <wsse:Algorithm Type="wsse:AlgEncryption"
          URI="http://www.w3.org/2001/04/xmlenc#3des-cbc" />
</wsse:Integrity>
```

By enclosing these assertions within the ExactlyOne operator (as previously illustrated), the requester knows to pick one of these encryption algorithms for ensuring message integrity because by defining these assertions, the provider is saying that it can accept encrypted messages in either the AES or 3DES format. The preferences of “100” and “1,” respectively, indicate the provider’s preference, with “100” indicating a higher preference for AES-based encryption.

WS-POLICYASSERTIONS

To really make a system of policy assertions and comparisons work, a standard set of assertions is needed. The WS-PolicyAssertion specification defines an initial set of assertions for simple policies, including the following:

TextEncoding— Defines supported character set encodings such as ASCII or UTF-8.
Language— Defines the supported and preferred natural languages for internationalization (may be dependent upon or related to supported character sets).
SpecVersion— Defines the supported specifications and their versions.

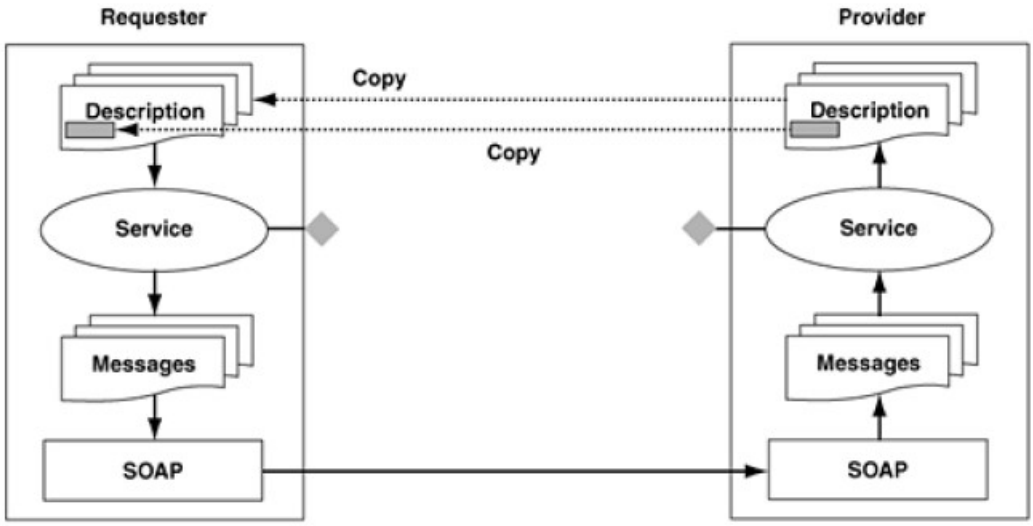
MessagePredicate— Defines a predicate expression that must evaluate to true as a way to flag acceptable messages

WS-POLICY ATTACHMENT

The purpose of the WS-Policy Attachment specification is to define how a policy is associated with an endpoint reference, Web service description, or UDDI entry.

WS-Addressing can be used to specify an endpoint reference with which to associate a policy assertion. WSDL extensibility points are used to associate WS-Policy assertions with a WSDL file.

The following figure. illustrates the association of policy information with the WSDL file. The shaded area represents the policy information. The policy information has to be available to the requester, just like the basic information about the Web service. The WS-Policy Attachments specification defines how the assertions are associated with the WSDL file, and the WS-Metadata Exchange can be used for discovering such referred policy schemas for a given WSDL file.



Policy information associated with WSDL is copied.

WS-Security

WS Security is a standard that addresses security when data is exchanged as part of a Web service. This is a key feature in SOAP that makes it very popular for creating web services.

Web services security includes several aspects:

Authentication—Verifying that the user is who she claims to be. A user's identity is verified based on the credentials presented by that user, such as providing password or biometric ID.

Authorization (or Access Control)—Granting access to specific resources based on an authenticated user's entitlements. Entitlements are defined by one or several attributes. An attribute is the property or characteristic of a user, for example, if "Marc" is the user, "conference speaker" is the attribute.

Confidentiality, privacy—Keeping information secret. Accesses a message, for example a Web service request or an email, as well as the identity of the sending and receiving parties in a

confidential manner. Confidentiality and privacy can be achieved by encrypting the content of a message and obfuscating the sending and receiving parties' identities.
Integrity, non repudiation—Making sure that a message remains unaltered during transit by having the sender digitally sign the message. A digital signature is used to validate the signature and provides non-repudiation. The timestamp in the signature prevents anyone from replaying this message after the expiration.

Web services security requirements also involve credential mediation (exchanging security tokens in a trusted environment), and service capabilities and constraints (defining what a Web service can do, under what circumstances).
Web services security requirements are supported by industry standards both at the transport level (Secure Socket Layer) and at the application level relying on XML frameworks.

Transport-level Security
Secure Socket Layer (SSL), otherwise known as Transport Layer Security (TLS), the Internet Engineering Task Force (IETF) officially standardized version of SSL, is the most widely used transport-level data-communication protocol providing:

- Authentication (the communication is established between two trusted parties).
- Confidentiality (the data exchanged is encrypted).
- Message integrity (the data is checked for possible corruption).
- Secure key exchange between client and server.

SSL provides a secure communication channel, however, when the data is not "in transit," the data is not protected. This makes the environment vulnerable to attacks in multi-step transactions. (SSL provides point-to-point security, as opposed to end-to-end security.)

Application-level Security
Application-level security complements transport-level security. Application-level security is based on XML frameworks defining confidentiality, integrity, authenticity; message structure; trust management and federation.
Data confidentiality is implemented by XML Encryption. XML Encryption defines how digital content is encrypted and decrypted, how the encryption key information is passed to a recipient, and how encrypted data is identified to facilitate decryption.
Data integrity and authenticity are implemented by XML Signature. XML Signature binds the sender's identity (or "signing entity") to an XML document. Signing and signature verification can be done using asymmetric or symmetric keys.
Signature ensures non-repudiation of the signing entity and proves that messages have not been altered since they were signed. Message structure and message security are implemented by SOAP and its security extension, WS-Security. WS-Security defines how to attach XML Signature and XML Encryption headers to SOAP messages. In addition, WS-Security provides profiles for 5 security tokens: Username (with password digest), X.509 certificate, Kerberos ticket, Security Assertion Markup Language (SAML) assertion, and REL (rights markup) document.
The SOAP envelope body includes the business payload, for example a purchase order, a financial document, or simply a call to another Web service. SAML is one of the most interesting security tokens because it supports both authentication and authorization. SAML is an open

framework for sharing security information on the Internet through XML documents. SAML includes 3 parts:

- SAML Assertion—How you define authentication and authorization information.
- SAML Protocol—How you ask (SAML Request) and get (SAML Response) the assertions you need.
- SAML Bindings and Profiles—How SAML assertions ride "on" (Bindings) and "in" (Profiles) industry-standard transport and messaging frameworks.

Advanced Messaging

Advanced messaging features include reliable messaging and extended message exchange patterns for event notification.

Reliable Messaging

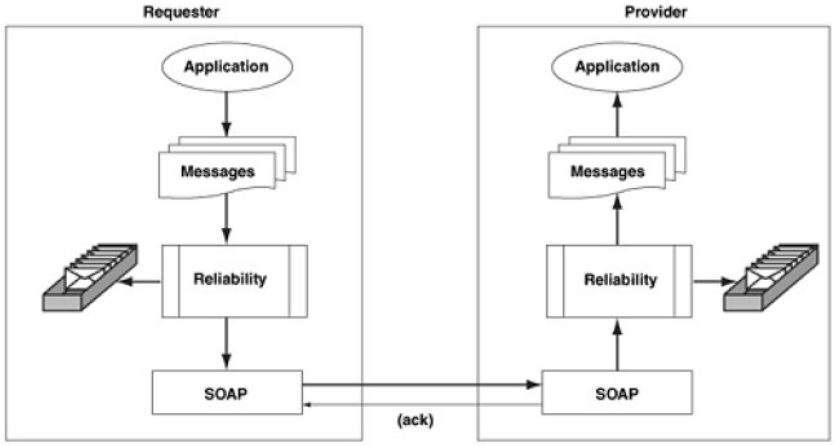
Reliability for Web services is defined independently of the transport as a series of SOAP messages exchanged within a group or sequence and some processing rules governing the use of acknowledgments and message numbers to ensure that all the messages are received, that (optionally) duplicates are eliminated, and that (optionally) message ordering is preserved.

Reliable messaging technology requires a piece of software infrastructure deployed on both ends of a connection. The reliable messaging agent handles errors in the transmission of messages from one computer to another over a (potentially unreliable) network. Typically, the agents are symmetrical implementations so that whatever is added to the processing of a message by the requester can be understood by the provider in the reciprocal processing needed to implement the reliability handshake.

The reliable messaging agent assigns a sequence ID to a group of related messages and message IDs to the individual messages within the group. The requester's agent marks the last message in the group, and the provider's agent returns an acknowledgment to indicate whether all messages in the group were received. If one or more message IDs is missing from the acknowledgment, the missing message (or messages) is re-sent until the provider's agent returns an acknowledgment containing all the message numbers in the group.

The sequence IDs and message IDs can also be used to prevent duplicate message processing and to require that messages be processed in the order they were sent, if that's important to the application. These numbers and associated processing options are included within SOAP headers. Reliable messaging mechanisms can be used with any SOAP MEP.

The following figure illustrates the basic architecture for reliable messaging: A reliability layer is introduced between the transport and the application that interprets and executes a series of message acknowledgments, typically working together with a persistence mechanism, to implement a reliability policy such as once-and-only-once guaranteed delivery. Acknowledgments let the SOAP requester know when the provider has received the message or set of messages and which ones were received.



Reliable messaging architecture.

Reliable messaging includes some or all of the following features:

- Guaranteed message delivery.
- Notification of message status.
- Duplicate elimination.
- Message ordering.

Notification

Notification provides a mechanism to publish messages about events so that subscribers can retrieve them independently of any relationship between provider and requester. Notification is a feature of many current distributed computing systems, including message-oriented middleware (MOM) and CORBA.

For example, a failure in a telephone-switching element generates an event that is passed to the notification system to post a message to a topic to which a network management console subscribes. When the message is received, the network management console knows to take corrective action. A connection is not required between the telephone-switching element and the management console in order to send the message notifying the management console of the failure.

Notification systems often include the use of an intermediary (such as an event broker or temporary storage channels) to which the message can be sent for later (i.e., asynchronous) retrieval by the consumer. Notification systems do not always require an intermediary, however, because the message providers can also support the temporary storage requirements necessary for an implementation.

For Web services, two notification specifications are proposed:

- **WS-Eventing—.** By BEA, Computer Associates, IBM, Microsoft, Sun, and Tibco Software.
- **WS-Notification—.** Submitted to the OASIS WS-Notification Technical Committee by Akami Technologies, IBM, Tibco Software, Fujitsu Software, SAP, Sonic Software, Computer Associates, Globus, and HP.

WS-Eventing is much smaller than WS-Notification (which is actually three specifications) and includes only very basic publish/subscribe technology. WS-Notification is part of a larger effort by IBM and others to provide messaging and resource management technologies for grid computing based on Web services.

WS-Eventing:

WS-Eventing is a simple specification that defines how to subscribe to a notification message, including a protocol for generating a confirmation response to a subscription request.

```
<wsa:Action>
        http://schemas.xmlsoap.org/ws/2004/01/eventing/Subscribe
</wsa:Action>
<wsa:ReplyTo>
      <wsa:Address>
              http://www.iona.com/Newcomer/StockOptions
     </wsa:Address>
</wsa:ReplyTo>
```

This simple example illustrates the WS-Eventing namespace wsa and the action to subscribe to an event (not shown) that, when it occurs, is published to Newcomer's event sink for stock option notices.

WS-Notification:

WS-Notification is the name for a family of three specifications that are being developed within the Web Services Resource Framework (WSRF) set of specifications.

The specifications include:

WS-BaseNotification- Defines interfaces for notification providers and requesters (called producers and consumers in the specification) and defines related MEPs.

WS-Topics—. Defines the topics mechanism, which is a way to categorize and organize subscriptions for different types of information (i.e., different message types). WS-Topics adds to WS-BaseNotification various use cases for publish/subscribe MEPs and specifies an XML model for metadata associated with pub/sub services.

WS-BrokeredNotification—. Defines the interface for a notification broker, which is an intermediary that manages the point-to-point and publish/subscribe MEPs.

WS-Transaction Management

The WS-CAF (Web Services Composite Application Framework) family's WS-Transaction Management (WS-TXM) specification defines three transaction protocols that can be plugged into WS-CF, including:

- A classic two-phase commit protocol called ACID.
- A compensation-based protocol called Long Running Action (LRA).
- A protocol specifically designed for use with automated business process flows called Business Process (BP).

ACID protocol

It includes heuristic error reporting to address the uncertainty state.

The ACID protocol in WS-TXM is also designed specifically to support interoperability across multiple variations of the two-phase commit protocol that exist in current and proposed systems.

Long Running Action (LRA) protocol

LRA is designed specifically for business interactions that occur over a long duration. An LRA is intended to reflect business interactions that are compensatable: All work performed within the

scope of an application must have a compensation action defined in order for it to be undone in the case of an error. The LRA protocol defines the triggers for compensation actions and the conditions under which those triggers are executed, but not the compensation actions themselves. For example, when a user reserves a seat on a flight, the airline may immediately book the seat and debit the user's charge card, relying on the fact that most of their customers actually buy the seats they book. The compensation action for this would be to un-book the seat and credit the user's charge card.

Business Process (BP) transaction protocol

In the Business Process (BP) transaction protocol defined in WS-TXM, all parties involved in a business process reside within business domains, which use business processes to perform work. In the Business Process (BP) transaction protocol defined in WS-TXM, all parties involved in a business process reside within business domains, which use business processes to perform work. Business process transactions are responsible for managing interactions between these domains. A business process (business-to-business interaction) is split into business tasks, and each task executes within a specific business domain. A business domain may itself be subdivided into other business domains (business processes) in a recursive manner.

Because each business domain typically provides different transaction protocols within different execution environments, a business task has to provide execution-specific error recovery. The controlling application may request that all of the business domains checkpoint their state such that they can either be consistently rolled back to that checkpoint by the application or restarted from the checkpoint in the event of a failure. The BP protocol is designed to interoperate across any combination of TP systems, whether a single resource manager system, multi-resource manager system, or more complex TP monitor or application server system, relying first on execution environment-specific recovery mechanisms and second on a coordinator-coordinator protocol to assemble execution environment information into an integrated, higher-level protocol to determine an overall outcome.