THE PRENTICE HALL SERVICE-ORIENTED COMPUTING SERIES FROM THOMAS ERL

"Service Oriented Architecture is a hot, but often misunderstood topic in IT today. Thomas articulately describes the concepts, specifications, and standards behind service orientation and Web Services. For enterprises adopting SOA, there is detailed advice for service-oriented analysis, planning, and design. This book is a must read!"
—Alex Lynch, Principal Consultant, Microsoft Enterprise Services

Service-Oriented Architecture

Concepts, Technology, and Design

PRENTICE HALL

Thomas Erl

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U. S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the U. S., please contact:

International Sales
international@pearsoned.com

Visit us on the Web: informit.com/ph

Library of Congress Number: 2005925019

ISBN-10 0-13-185858-0
ISBN-13 978-0-13-185858-9

Text printed in the United States on recycled paper at R.R. Donnelley in Crawfordsville, Indiana.
Ninth printing, February 2009

Preface

Authoring this book involved nearly a year of writing, research, and staying on top of a subject matter that is constantly expanding its reach and importance. Although the majority of the chapters focus on service-oriented architecture from a vendor-neutral perspective, achieving an accurate representation of this perspective required that I spend a great deal of time evaluating SOA support in all primary vendor platforms. As part of this research stage I spoke with more than a hundred senior IT professionals, either through interviews or through my work as an awards judge evaluating platform submissions.

One of the most interesting facets of this project has been in defining service-orientation within the context of Web services. While studying the individual parts of what constitutes service-orientation as a paradigm, I came to realize just how many of its roots lie in past innovations. Yet at the same time, it is distinct, blending traditional and new concepts in support of a unique architectural model.

Despite its apparent "newness," SOA, on a fundamental level, is based on a very old and established school of thought. Service-orientation, as a means of separating things into independent and logical units, is a very common concept. As I progressed through these chapters, I began to notice this more often in everyday life. Items, people, organizations we come into contact with either offer some form of service or participate in performing a service. Once applied to technology architecture, though, service-orientation is concerned with a specific part of our service-oriented world: business automation.

Competitive business climates demand that corporations minimize redundant effort and maximize the expediency with which strategic goals can be achieved. Inefficient organizations that consistently waste resources are bound to fall behind. The manner in which an organization automates its business is a critical factor in determining the level of efficiency at which it operates and, ultimately, the extent of success it attains in its ventures.

This is what makes SOA so valuable. By shaping automation logic through service-orientation, existing investments can be leveraged, business intelligence can be accurately expressed, and inherent automation agility can be achieved. When coupled with the Web services technology platform, SOA offers a significant and real benefit potential that can transform the technology and outlook of an organization. My goal for this book is to help you explore, understand, and realize this potential.

Acknowledgments

While writing this book I was blessed with a strong team of technical reviewers and superior editorial, production, and marketing professionals. My thanks to all of you for your tireless efforts. A special thanks to my family for their patience and unwavering support.

Chapter 7

Web Services and Contemporary SOA (Part II: Advanced Messaging, Metadata, and Security)

In Chapter 6 we established a series of composition and activity management concepts, each with a different scope and purpose, but all somewhat related within the context of composable SOA. Those initial concepts are complemented by additional WS-* extensions that govern specific areas of the SOAP messaging framework, the creation and exchange of metadata, and the introduction of message-level security. (Figure 7.1 introduces the individual concepts and shows how they typically inter-relate.)

As we explore the various extensions in this chapter, it becomes increasingly clear that SOAP messaging is the lifeblood of contemporary service-oriented architecture. It realizes not only the delivery of application data, but also the composable nature of SOA. The innovation of SOAP headers accounts for almost all of the features covered in Chapters 6 and 7.

To demonstrate common concepts, this chapter borrows terms provided by the following current Web services specifications:

- WS-Addressing
- WS-ReliableMessaging
- WS-Policy Framework (including WS-PolicyAttachments and WS-PolicyAssertions)
- WS-MetadataExchange
- WS-Security (including XML-Encryption, XML-Signature, and SAML)
- WS-Notification Framework (including WS-BaseNotification, WS-Topics, and WS-BrokeredNotification)
- WS-Eventing

As with Chapter 6, we only explore concepts related to WS-* extensions in this chapter. Language element descriptions and examples for the first five specifications in the preceding list are provided in Chapter 17.

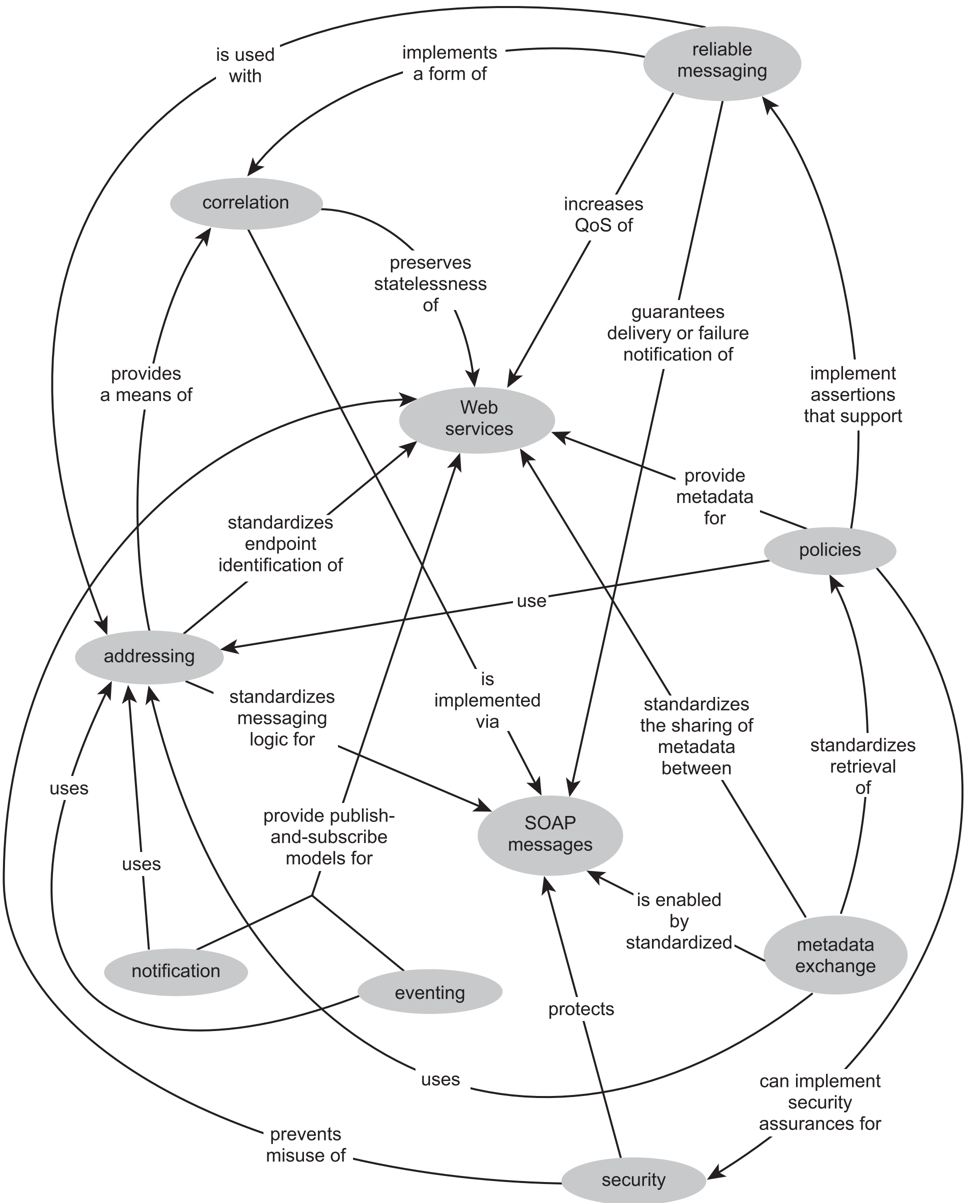

Figure 7.1
Specifications and concepts covered in this chapter.

NOTE

Markup code examples for WS-Eventing and the WS-Notification framework are not provided. These two specifications provide different languages that cover much of the same ground and are more relevant to the subject matter in this book on a conceptual level.

In Plain English sections

This chapter also contains *In Plain English* sections for every primary concept discussed. Note that these intentionally simplistic analogies continue where they left off in Chapter 6.

How case studies are used: Several of the examples in Chapter 6 are revisited here as we take a closer look at how interaction among specific TLS Web services is affected by the new concepts introduced in this chapter.

7.1 Addressing

What addressing brings to SOAP messaging is much like what a waybill brings to the shipping process. Regardless of which ports, warehouses, or delivery stations a package passes through en route to its ultimate destination, with a waybill attached to it, everyone it comes into contact with knows:

- where it's coming from
- the address of where it's supposed to go
- the specific person at the address who is supposed to receive it
- where it should go if it can't be delivered as planned

The WS-Addressing specification implements these addressing features (Figure 7.2) by providing two types of SOAP headers (explained shortly). Though relatively simple in nature, these addressing extensions are integral to SOA's underlying messaging mechanics. Many other WS-* specifications implicitly rely on the use of WS-Addressing.

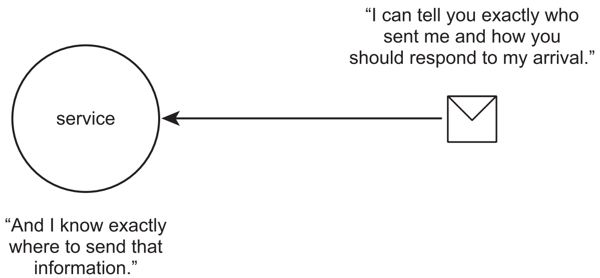

Figure 7.2
Addressing turns messages into autonomous units of communication.

NOTE
For an overview of the WS-Addressing language, see the *WS-Addressing language basics* section in Chapter 17.

IN PLAIN ENGLISH

As our car washing company grows, so do the administration duties. Every week Chuck reviews the mail and takes care of necessary paperwork. This week he receives two letters: one from our insurance company and the other from the tax office.

The first letter includes our renewed insurance policy statement, along with an invoice for another year of coverage. The "from" address on this letter is simply the name and location of the insurance company's head office. The enclosed statement contains a letter written by our account representative, outlining some of the changes in this year's policy and requesting that we mail our check directly to him. Chuck therefore encloses our payment in an envelope with a "to" address that includes an "attention" line stating that this letter should be delivered directly to the account representative.

The next letter contains another bill. This time, it's a tax statement accompanied by a letter of instruction and two return envelopes. According to the instructions, we are to use the first envelope (addressed to the A/R office) to mail a check if we are paying the full amount owing. If we cannot make a full payment, we need to use the second envelope (addressed to the collections department) to send whatever funds we have.

These scenarios, in their own crude way, demonstrate the fundamental concepts of endpoint references and message information headers, which are explained in the following sections.

7.1.1 Endpoint references

Early on in this book we established that the loosely coupled nature of SOA was implemented through the use of service descriptions. In other words, all that is required for a service requestor to contact a service provider is the provider's WSDL definition. This document, among other things, supplies the requestor with an address at which the provider can be contacted. What if, though, the service requestor needs to send a message to a specific instance of a service provider? In this case, the address provided by the WSDL is not sufficient.

Traditional Web applications had different ways of managing and communicating session identifiers. The most common approach was to append the identifier as a query string parameter to the end of a URL. While easy to develop, this technique resulted in application designs that lacked security and were non-standardized.

The concept of addressing introduces the *endpoint reference*, an extension used primarily to provide identifiers that pinpoint a particular instance of a service (as well as supplementary service metadata). The endpoint reference is expected to be almost always dynamically generated and can contain a set of supplementary properties.

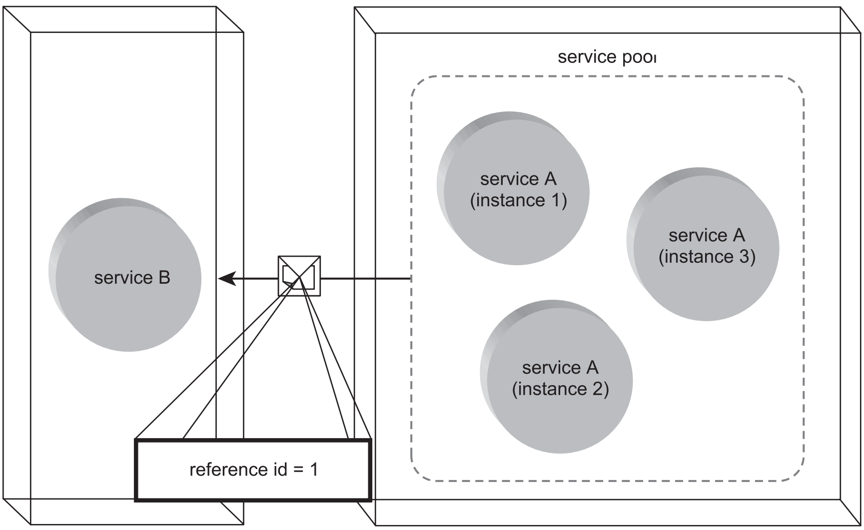

Figure 7.3
A SOAP message containing a reference to the instance of the service that sent it.

An endpoint reference consists of the following parts:

- *address*—The URL of the Web service.
- *reference properties*—A set of property values associated with the Web service instance. (In our previous *In Plain English* example, the "attention" line used in the first scenario is representative of the reference ID property.)
- *reference parameters*—A set of parameter values that can be used to further interact with a specific service instance.
- *service port type and port type*—Specific service interface information giving the recipient of the message the exact location of service description details required for a reply.
- *policy*—A WS-Policy compliant policy that provides rules and behavior information relevant to the current service interaction (policies are explained later in this chapter).

Additional parts exist, which mostly identify corresponding WSDL information. With the exception of the address, all parts are optional.

7.1.2 Message information headers

In the previous chapter we covered the various primitive message exchange patterns of which complex activities are comprised. These MEPs have predictable characteristics that can ease the manner in which Web services are designed but also can limit the service interaction scenarios within which they participate.

In sophisticated service-oriented solutions, services often require the flexibility to break a fixed pattern. For example, they may want to dynamically determine the nature of a message exchange. The extensions provided by WS-Addressing were broadened to include new SOAP headers that establish message exchange-related characteristics within the messages themselves. This collection of standardized headers is known as the *message information* (or *MI*) headers (Figure 7.4).

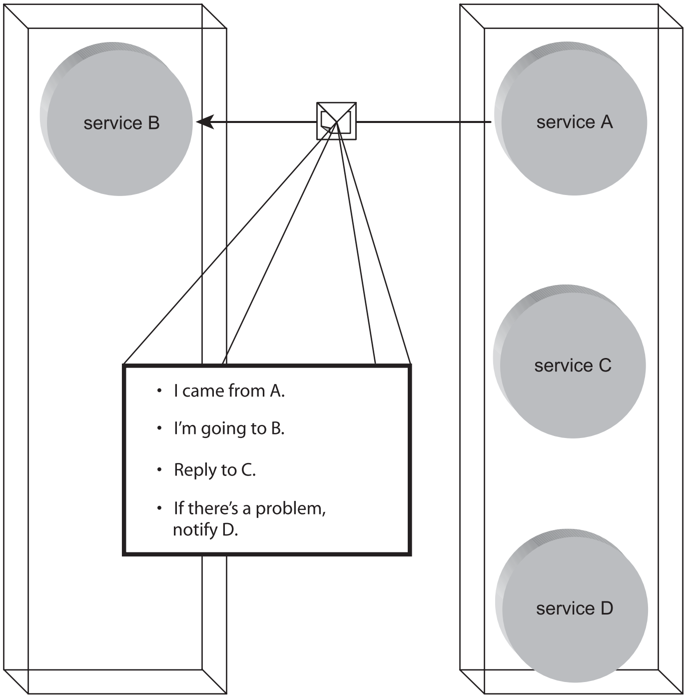

Figure 7.4
A SOAP message with message information headers specifying exactly how the recipient service should respond to its arrival.

The MI headers provided by WS-Addressing include:

- *destination*—The address to which the message is being sent.
- *source endpoint*—An endpoint reference to the Web service that generated the message.
- *reply endpoint*—This important header allows a message to dictate to which address its reply should be sent.
- *fault endpoint*—Further extending the messaging flexibility is this header, which gives a message the ability to set the address to which a fault notification should be sent.
- *message id*—A value that uniquely identifies the message or the retransmission of the message (this header is required when using the reply endpoint header).

- *relationship*—Most commonly used in request-response scenarios, this header contains the message id of the related message to which a message is replying (this header also is required within the reply message).
- *action*—A URI value that indicates the message's overall purpose (the equivalent of the standard SOAP HTTP action value).

(Also of interest is the fact that the WS-Addressing specification provides an anonymous URI that allows MI headers to intentionally contain an invalid address.)

Outfitting a SOAP message with these headers further increases its position as an independent unit of communication. Using MI headers, SOAP messages now can contain detailed information that defines the messaging interaction behavior of the service in receipt of the message. The net result is standardized support for the use of unpredictable and highly flexible message exchanges, dynamically creatable and therefore adaptive and responsive to runtime conditions.

7.1.3 Addressing and transport protocol independence

Historically, many of the details pertaining to how a unit of communication arrives at point B after it is transmitted from point A was left up to the individual protocols that controlled the transportation layer. While this level of technology-based abstraction is convenient for developers, it also leads to restrictions as to how communication between two units of processing logic can be achieved.

The standardized SOAP headers introduced by WS-Addressing remove much of this protocol-level dependence. These headers put the SOAP message itself in charge of its own destiny by further increasing its ability to act as a standalone unit of communication.

7.1.4 Addressing and SOA

Addressing achieves an important low-level, transport standardization within SOA, further promoting open standards that establish a level of transport technology independence (Figure 7.5). The use of endpoint references and MI headers deepens the intelligence embedded into SOAP messages, increasing message-level autonomy.

Empowering a message with the ability to self-direct its payload, as well as the ability to dictate how services receiving the message should behave, significantly increases the potential for Web services to be intrinsically interoperable. It places the task-specific logic into the message and promotes a highly reusable and generic service design standard that also facilitates the discovery of additional service metadata.

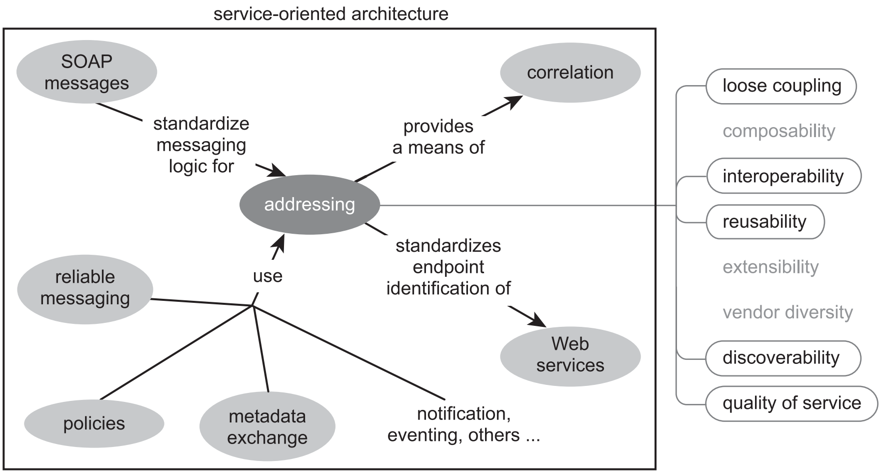

Figure 7.5
Addressing relating to other parts of SOA.

Further, the use of MI headers increases the range of interaction logic within complex activities and even encourages this logic to be dynamically determined. This, however, can be a double-edged sword. Even though MI headers can further increase the sophistication of service-oriented applications, their misuse (or overuse) can lead to some wildly creative and complex service activities.

Finally, by supporting the referencing of service instances, SOAs can be scaled in a standardized manner, without the need to resort to custom or proprietary application designs (scalability is a key QoS contribution). Having stated that, it should be pointed out that by providing functionality that enables communication with service instances, WS-Addressing indirectly supports the creation of stateful services. This runs contrary to the common service-orientation principle of statelessness (as explained in Chapter 8) and emphasizes the need for this feature to be applied in moderation.

CASE STUDY

In the previous chapter we provided several examples that explained the steps behind the vendor invoice submission process. One of these steps required that, upon receiving an invoice from a vendor, the TLS Accounts Payable Service interacts with the TLS Vendor Profile Service to have the received invoice validated against vendor account information already on file.

Due to the volume of invoice submissions received by TLS, there can be, at any given time, multiple active instances of the Accounts Payable Service. Therefore, as part of the message issued by the Accounts Payable Service to the Vendor Profile Service, a SOAP header providing a reference id is included. This identifier represents the current instance of the Accounts Payable Service and is used by the Vendor Profile Service to locate this instance when it is ready to respond with the validation information (Figure 7.6).

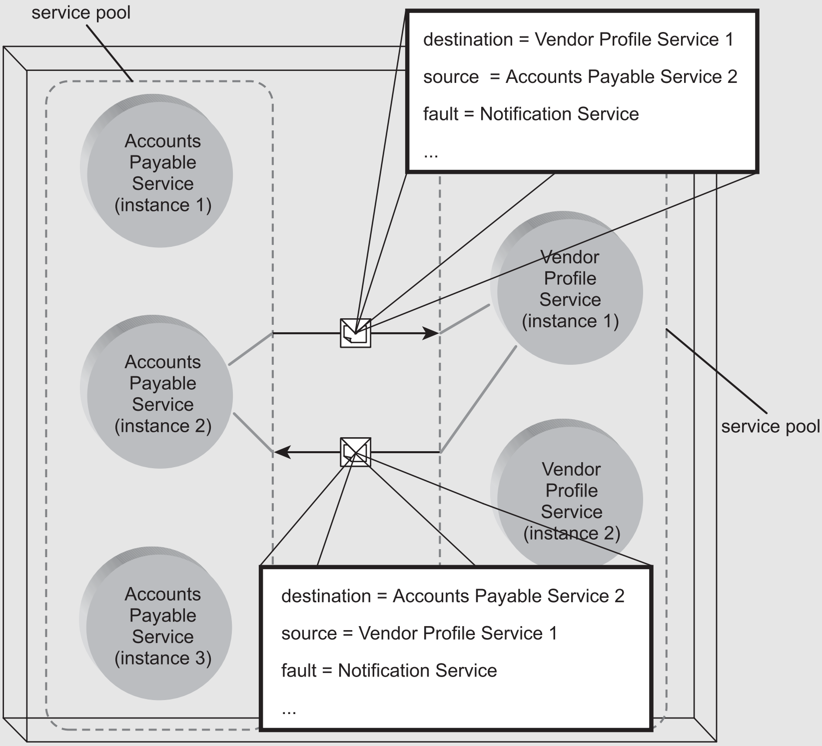

Figure 7.6
Separate service instances communicating using endpoint references and MI headers across two pools of Web services within TLS.

SUMMARY OF KEY POINTS

- Addressing extensions, as implemented by the WS-Addressing specification, introduce two important concepts: endpoint references and message information headers.
- Endpoint references provide a standardized means of identifying a specific instance of a Web service.
- Message information headers add message exchange properties to a specific message, conveying interaction semantics to recipient services.
- Though simple in comparison to other WS-* specifications, WS-Addressing inserts a powerful layer of messaging autonomy within a service-oriented architecture.

7.2 Reliable messaging

The benefits of a loosely coupled messaging framework come at the cost of a loss of control over the actual communications process. After a Web service transmits a message, it has no immediate way of knowing:

- whether the message successfully arrived at its intended destination
- whether the message failed to arrive and therefore requires a retransmission
- whether a series of messages arrived in the sequence they were intended to

Reliable messaging addresses these concerns by establishing a measure of quality assurance that can be applied to other activity management frameworks (Figure 7.7).

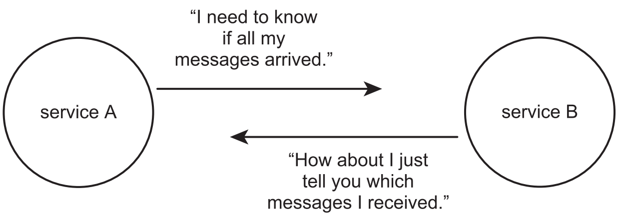

Figure 7.7

Reliable messaging provides a guaranteed notification of delivery success or failure.

WS-ReliableMessaging provides a framework capable of guaranteeing:

- that service providers will be notified of the success or failure of message transmissions
- that messages sent with specific sequence-related rules will arrive as intended (or generate a failure condition)

Although the extensions introduced by reliable messaging govern aspects of service activities, the WS-ReliableMessaging specification is different from the activity management specifications we discussed in Chapter 6. Reliable messaging does not employ a coordinator service to keep track of the state of an activity; instead, all reliability rules are implemented as SOAP headers within the messages themselves.

NOTE

Chapter 17 provides an introduction to the WS-ReliableMessaging language in the *WS-ReliableMessaging language basics* section.

IN PLAIN ENGLISH

In the last chapter's *Choreography* section we explained how our car wash had formed an alliance with the car wash located on the other side of the highway. Part of our arrangement was to share part-time workers during peak hours.

One of the workers that joined our team is named George. Though good at his job, George has a bad memory. When we request that workers from the other side walk over to help us out, we always are warned when one of those workers is George.

The walk from the other gas station is about one kilometer. Sometimes George forgets the way and gets lost. We therefore put a system in place where we agree to call the other company to tell them how many workers have arrived. If it's not equal to the number of workers they actually sent, it's usually because George has gone missing again.

Our system of calling the other company to acknowledge the receipt of the workers and to report any missing workers builds an element of reliability into our resource sharing arrangement.

7.2.1 RM Source, RM Destination, Application Source, and Application Destination

WS-ReliableMessaging makes a distinction between the parts of a solution that are responsible for initiating a message transmission and those that actually perform the transmission. It further assigns specific descriptions to the terms "send," "transmit," "receive," and "deliver," as they relate differently to these solution parts. These differentiations are necessary to abstract the reliable messaging framework from the overall SOA.

An *application source* is the service or application logic that *sends* the message to the *RM source*, the physical processor or node that performs the actual wire *transmission*. Similarly, the *RM destination* represents the target processor or node that *receives* the message and subsequently *delivers* it to the *application destination* (Figure 7.8).

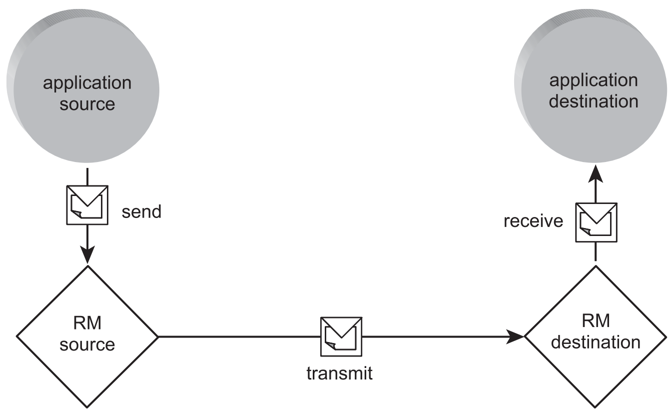

Figure 7.8
An application source, RM source, RM destination, and application destination.

7.2.2 Sequences

A *sequence* establishes the order in which messages should be delivered. Each message that is part of a sequence is labeled with a *message number* that identifies the position of the message within the sequence. The final message in a sequence is further tagged with a *last message* identifier.

7.2.3 Acknowledgements

A core part of the reliable messaging framework is a notification system used to communicate conditions from the RM destination to the RM source. Upon receipt of the message containing the last message identifier, the RM destination issues a *sequence acknowledgement* (Figure 7.9). The acknowledgement message indicates to the RM source which messages were received. It is up to the RM source to determine if the messages received are equal to the original messages transmitted. The RM source may retransmit any of the missing messages, depending on the delivery assurance used (see following section).

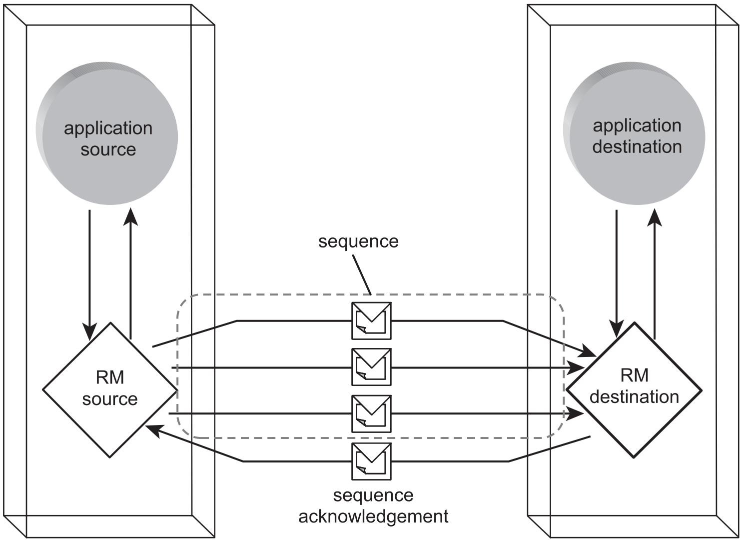

Figure 7.9
A sequence acknowledgement sent by the RM destination after the successful delivery of a sequence of messages.

An RM source does not need to wait until the RM destination receives the last message before receiving an acknowledgement. RM sources can request that additional acknowledgements be transmitted at any time by issuing *request acknowledgements* to RM destinations (Figure 7.10). Additionally, RM destinations have the option of transmitting *negative acknowledgements* that immediately indicate to the RM source that a failure condition has occurred (Figure 7.11).

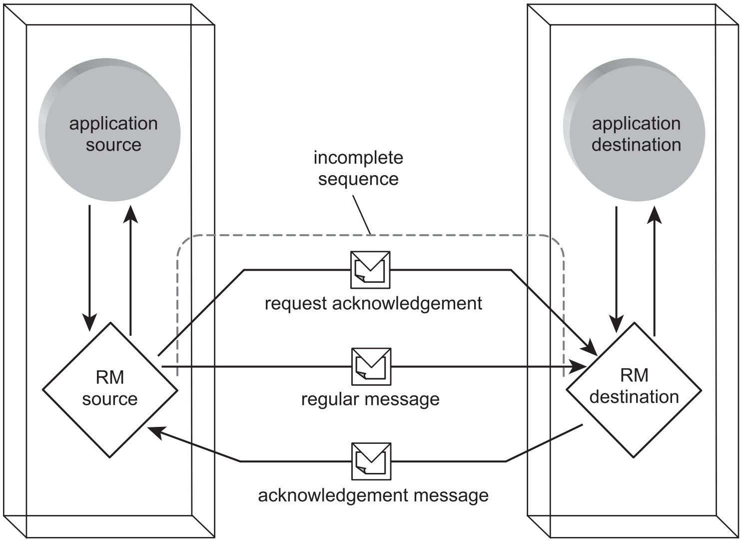

Figure 7.10

A request acknowledgement sent by the RM source to the RM destination, indicating that the RM source would like to receive an acknowledgement message before the sequence completes.

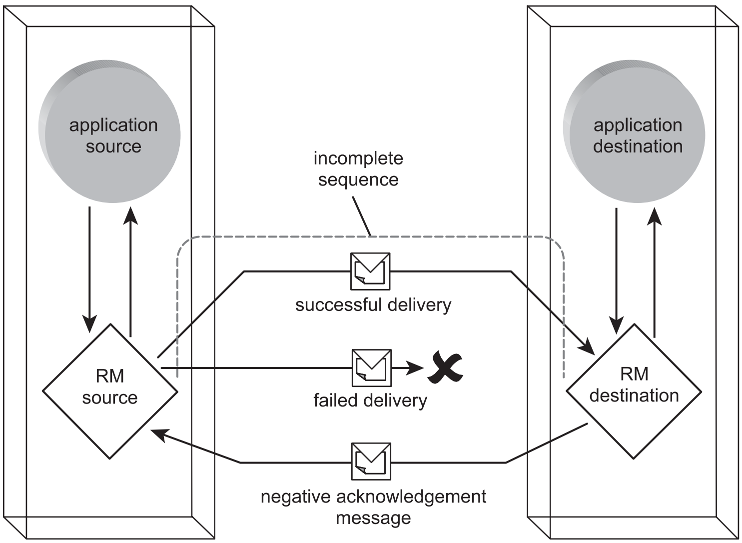

Figure 7.11

A negative acknowledgement sent by the RM destination to the RM source, indicating a failed delivery prior to the completion of the sequence.

7.2.4 Delivery assurances

The nature of a sequence is determined by a set of reliability rules known as *delivery assurances*. Delivery assurances are predefined message delivery patterns that establish a set of reliability policies.

The following delivery assurances are supported:

The *AtMostOnce* delivery assurance promises the delivery of one or zero messages. If more than one of the same message is delivered, an error condition occurs (Figure 7.12).

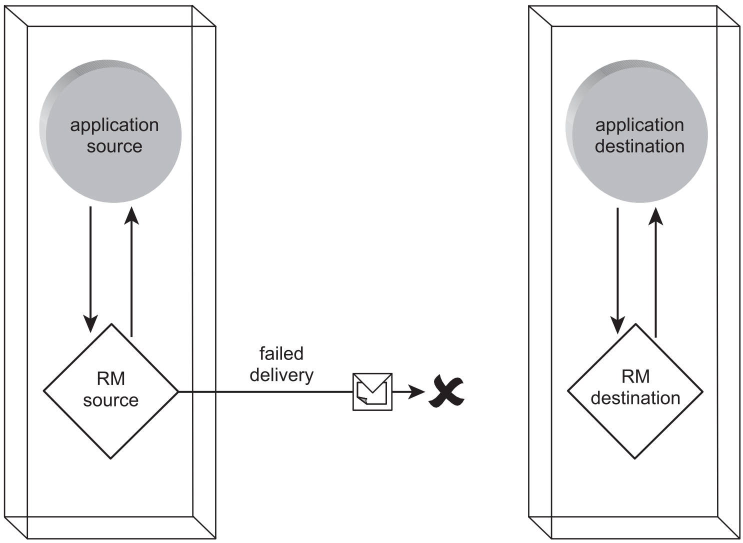

Figure 7.12
The AtMostOnce delivery assurance.

The *AtLeastOnce* delivery assurance allows a message to be delivered once or several times. The delivery of zero messages creates an error condition (Figure 7.13).

The *ExactlyOnce* delivery assurance guarantees that a message only will be delivered once. An error is raised if zero or duplicate messages are delivered (Figure 7.14).

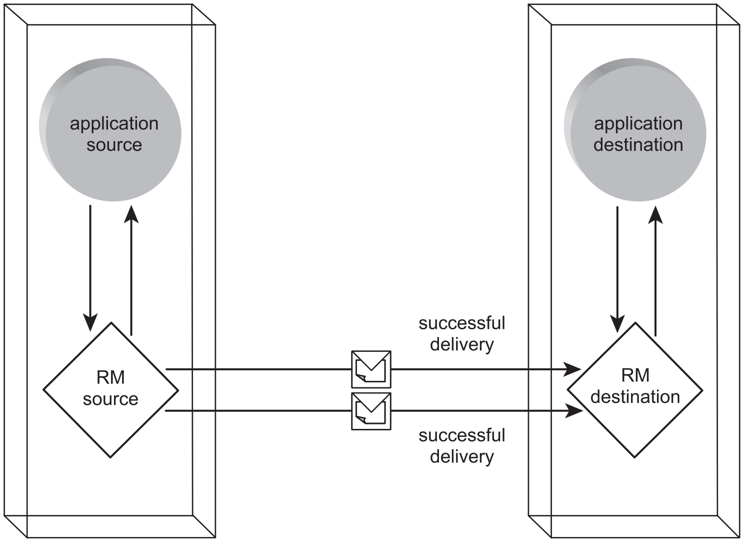

Figure 7.13
The AtLeastOnce delivery assurance.

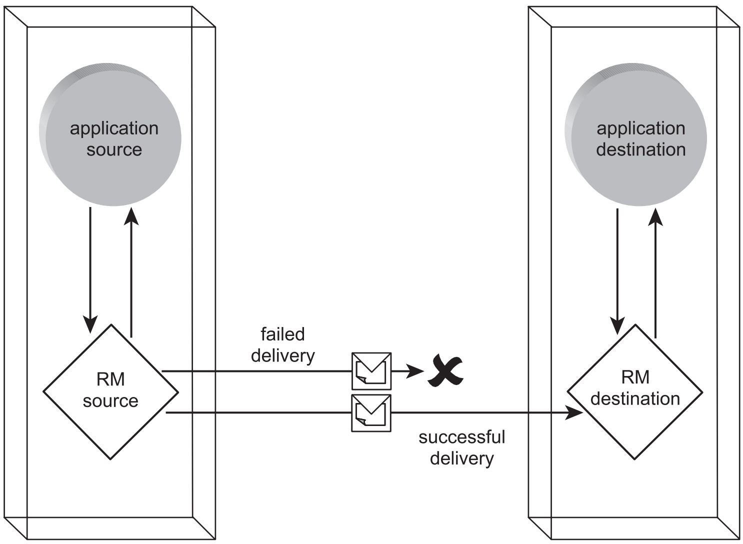

Figure 7.14
The ExactlyOnce delivery assurance.

The *InOrder* delivery assurance is used to ensure that messages are delivered in a specific sequence (Figure 7.15). The delivery of messages out of sequence triggers an error. Note that this delivery assurance can be combined with any of the previously described assurances.

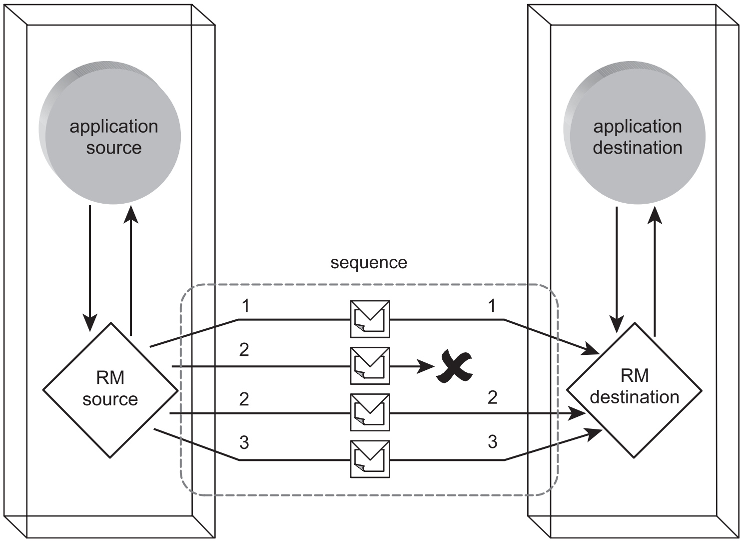

Figure 7.15
The InOrder delivery assurance.

7.2.5 Reliable messaging and addressing

WS-Addressing is closely tied to the WS-ReliableMessaging framework. In fact, it's interesting to note that the rules around the use of the WS-Addressing message id header were altered specifically to accommodate the WS-ReliableMessaging specification. Originally, message id values always had to be unique, regardless of the circumstance. However, the delivery assurances supported by WS-ReliableMessaging required the ability for services to retransmit identical messages in response to communication errors. The subsequent release of WS-Addressing, therefore, allowed retransmissions to use the same message ID.

7.2.6 Reliable messaging and SOA

Reliable messaging brings to service-oriented solutions a tangible quality of service (Figure 7.16). It introduces a flexible system that guarantees the delivery of message sequences supported by comprehensive fault reporting. This elevates the robustness of

SOAP messaging implementations and eliminates the reliability concerns most often associated with any messaging frameworks.

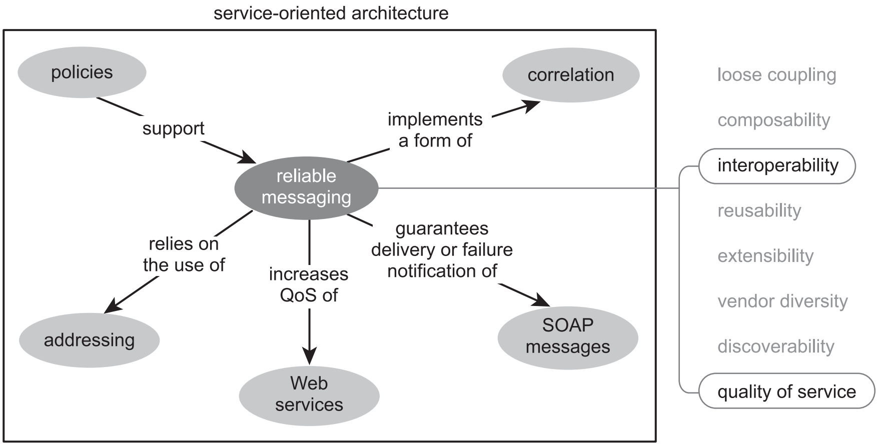

Figure 7.16
Reliable messaging relating to other parts of SOA.

By increasing the delivery quality of SOAP messages, reliable messaging increases the quality of cross-application communication channels as well. The limitations of a messaging framework no longer inhibit the potential of establishing enterprise-level integration.

CASE STUDY

To accommodate their existing accounting practices, RailCo sometimes prefers to issue bulk, month-end invoice submissions. TLS has other vendors that require this option and therefore accepts these forms of bulk submissions—but under the condition that they must be transmitted as part of the same sequence. This gives TLS the ability to issue an acknowledgement that communicates which of the invoice messages were successfully received.

RailCo complies with this requirement and enhances its existing Invoice Submission Service to package invoices in SOAP messages that support reliable messaging extensions (Figure 7.17).

The first submitted batch consists of 15 invoices. Much to RailCo's dismay, upon transmitting the last message in the sequence, TLS issues an acknowledgement message indicating that only 11 of the 15 invoice messages were actually received. In preparation for subsequent bulk submissions, RailCo extends its

Invoice Submission Service to issue an acknowledgement request message after every second invoice message sent as part of a sequence. This allows RailCo to better monitor and respond to failed delivery attempts.

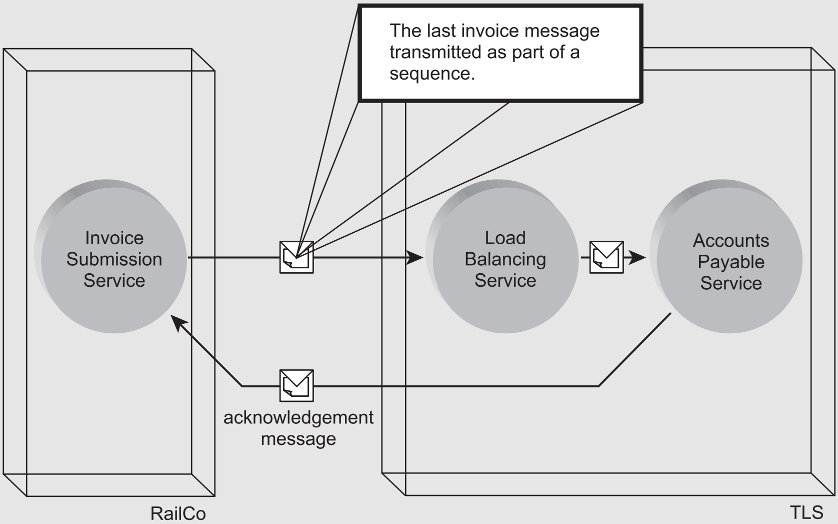

Figure 7.17
After transmitting a series of invoice messages, the last message within the sequence triggers the issuance of an acknowledgement message by TLS.

NOTE

The passive Load Balancing Service displayed in Figure 7.17 does not verify or process reliability conditions. Messages are simply passed through to the destination Accounts Payable Service.

SUMMARY OF KEY POINTS

- WS-ReliableMessaging establishes a framework that guarantees the delivery of a SOAP message or the reporting of a failure condition.
- The key parts of this framework are a notification system based on the delivery of acknowledgement messages and a series of delivery assurances that provide policies comprised of reliability rules.
- WS-ReliableMessaging is closely associated with the WS-Addressing and WS-Policy specifications.
- Reliable messaging significantly increases SOA's quality of service level and broadens its interoperability potential.

7.3 Correlation

One of the fundamental requirements for exchanging information via Web services is the ability to persist context and state across the delivery of multiple messages. Because a service-oriented communications framework is inherently loosely coupled, there is no intrinsic mechanism for associating messages exchanged under a common context or as part of a common activity. Even the execution of a simple request-response message exchange pattern provides no built-in means of automatically associating the response message with the original request.

Correlation addresses this issue by requiring that related messages contain some common value that services can identify to establish their relationship with each other or with the overall task they are participating in (Figure 7.18). The specifications that realize this simple concept provide different manners of implementation. We therefore dedicate the following section to explaining what correlation is and comparing how it is implemented by some of the WS-* extensions we've covered so far.

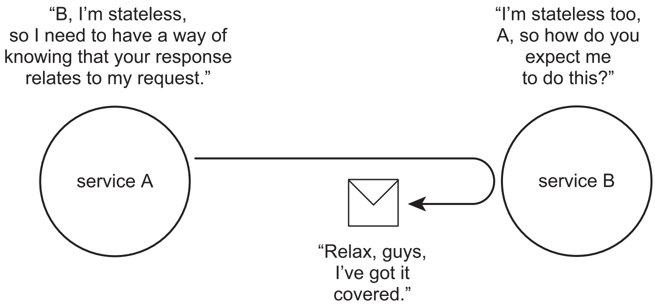

Figure 7.18
Correlation places the association of message exchanges into the hands of the message itself.

NOTE

For a look at how correlation typically is implemented as part of SOAP headers, see the examples provided in the *WS-Addressing language basics* section in Chapter 17.

IN PLAIN ENGLISH

To encourage repeat business, we introduce a promotion where, after ten visits to our car wash, your eleventh visit is free. We implement this promotion through the use of a punch card. Every time a customer drives in, we punch the driver's card. This card associates the current visit with all of the previous visits. Essentially, the punch card provides us with a form of correlation. Without it we would have a very hard time remembering which customers had visited us before.

7.3.1 Correlation in abstract

To establish a neutral point of reference, let's start with a very basic description of correlation without any reference to its implementation. In tightly bound communication frameworks the issue of correlated units of communication (individual transmissions) rarely arose. The technology that enabled tightly bound communication between components, databases, and legacy applications typically established an active connection that persisted for the duration of a given business activity (or longer). Because the connection remained active, context was inherently present, and correlation between individual transmissions of data was intrinsically managed by the technology protocol itself.

Things change dramatically when you fiddle with the coupling, however. When one stateless service sends a message to another, it loses control of the message and preserves no context of the activity in which the message is participating. It is up to the message to introduce the concept of correlation to provide services with the ability to associate a message with others.

This is achieved by embedding a value in the message that is propagated to all related messages. When a service processes a message and locates this value, it can establish a form of context, in that it can be used to associate this message with others. The nature of the context can vary. For example, a message could be part of a simple exchange activity, an atomic transaction, or a long running orchestration.

Let's now take a look at how correlation is achieved within some of the composition environments we covered in Chapter 6 and the messaging extensions discussed so far in this chapter.

7.3.2 Correlation in MEPs and activities

Because they are generic and non-business-specific in nature, MEPs and activities have no predefined notion of correlation. They are simple, conceptual building blocks incorporated and assembled by either custom-developed solutions that employ custom correlation identifiers and related processing logic or by specifications that impose proprietary forms of correlation (as described next).

7.3.3 Correlation in coordination

The context management framework provided by WS-Coordination establishes a sophisticated mechanism for propagating identifiers and context information between services. A separate activation service is responsible for creating new activities and subsequently generating and distributing corresponding context data. Services can forward this information to others that can use it to register for participation in the activity.

While context management uses a correlation-type identifier to uniquely represent a specific activity context, it goes well beyond correlation features to provide a comprehensive context management framework that can be leveraged through activity protocols, such as those supplied by the WS-AtomicTransaction and WS-BusinessActivity extensions.

7.3.4 Correlation in orchestration

WS-BPEL orchestrations need to concern themselves with the correlation of messages between process and partner services. This involves the added complexity of representing specific process instances within the correlation data. Further complicating this scenario is the fact that a single message may participate in multiple contexts, each identified by a separate correlation value.

To facilitate these requirements, the WS-BPEL specification defines specific syntax that allows for the creation of extensible *correlation sets*. These message properties can be dynamically added, deleted, and altered to reflect a wide variety of message exchange scenarios and environments.

7.3.5 Correlation in addressing

WS-Addressing's message id and relationship MI headers provide inherent correlation abilities, which can be leveraged by many composition and messaging extensions.

7.3.6 Correlation in reliable messaging

Every message that participates in a WS-ReliableMessaging sequence carries sequence information with it. This data consists of a sequence identifier that represents the series of messages required to follow the messaging rules of the sequence, along with a message identifier that identifies how the current message fits into the overall sequence. As a whole, this information can be considered correlation-related. However, its primary purpose is to support the enforcement of reliability rules.

7.3.7 Correlation and SOA

Correlation is a key contributor to preserving service autonomy and statelessness. Though simple by nature, the ability to tie messages together without requiring that services somehow manage this association is an important function of correlation, primarily because of how common message associations are in enterprise SOAs.

CASE STUDY

The PO Submission Process we described in Chapter 6 consists of a complex activity involving the TLS Purchase Order and the RailCo Order Fulfillment Services. In our previous examples we explained how the path of the PO message can be determined and extended dynamically at runtime and how it spans multiple services.

For each service that receives the PO message to understand the context under which it should process the message contents, it needs to be able to differentiate the message from others. It accomplishes this by associating a unique value with the message. In this case, the message identifier used is a value partially auto-generated and partially derived from the message PO number.

SUMMARY OF KEY POINTS

- Correlation is a required part of any SOA, as it enables the persistence of activity context across multiple message exchanges, while preserving the loosely coupled nature of service-oriented solutions.
- WS-* specifications implement correlation in different ways, but many specifications increasingly are relying on WS-Addressing for a form of standardized correlation.
- Even though values from a message's content can be used for correlation purposes, SOAP headers are the most common location for correlation identifiers.
- Correlation is an essential part of messaging within SOA, as it preserves service statelessness and (to an extent) supports message autonomy.

7.4 Policies

We now take a bit of a leap from the advanced messaging part of this chapter over to the WS-* extensions that provide enhanced metadata features for Web services.

Every automated business task is subject to rules and constraints. These characteristics trickle down to govern the behavior of the underlying services that automate the task.

The source of these restrictions could be:

- actual business-level requirements
- the nature of the data being exchanged
- organizational security measures

Further, every service and message has unique characteristics that may be of interest to other services that cross its path.

Examples include:

- behavioral characteristics
- preferences
- technical limitations
- quality of service (QoS) characteristics

Services can be outfitted with publicly accessible metadata that describes properties such as the ones listed here. This information is housed in a *policy* (Figure 7.19).

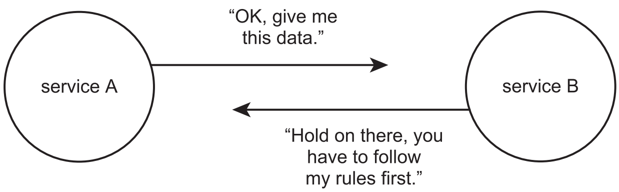

Figure 7.19
Policies can express a variety of service properties, including rules.

The use of policies allows a service to express various characteristics and preferences and keeps it from having to implement and enforce rules and constraints in a custom manner. It adds an important layer of abstraction that allows service properties to be independently managed.

> **NOTE**
>
> This section focuses on the design of policies for use with Web services. It is worth noting that polices can be attached to additional types of Web resources.

> **IN PLAIN ENGLISH**
>
> The first thing drivers see when they pull up to our operation is a sign that explains a few things about the car wash.
>
> The sign lists three specific points:
>
> - After driving to the car washing area, turn the engine off and exit the car.
> - Our power washing equipment can be very loud. Beware.
> - We recommend that you wait inside the gas station until the car wash has completed.
>
> The first point is a rule that customers must follow before the car washing process can begin. The second is an informational statement explaining a behavioral characteristic of the car wash. The final point indicates a preference of ours (it is safer for customers and easier for us if they stay out of the way of the workers). Each of these items expresses part of an overall policy.

7.4.1 The WS-Policy framework

The WS-Policy framework establishes extensions that govern the assembly and structure of policy description documents (Figure 7.20), as well as the association of policies to Web resources. This framework is comprised of the following three specifications:

- WS-Policy
- WS-PolicyAttachments
- WS-PolicyAssertions

Note also that the WS-Policy framework forms part of the WS-Security framework. Specifically, the WS-SecurityPolicy specification defines a set of policy assertions intended for use with WS-Security (introduced later in this chapter).

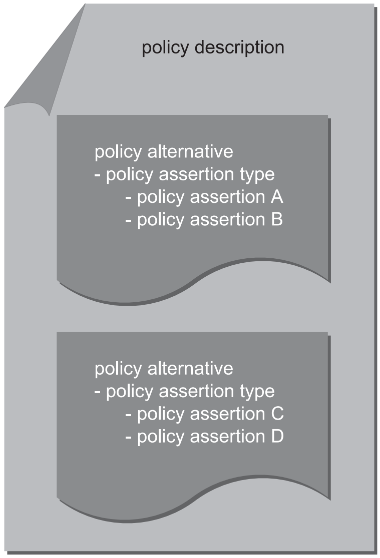

Figure 7.20
The basic structure of a policy description.

Policies can be programmatically accessed to provide service requestors with an understanding of the requirements and restrictions of service providers at runtime. Alternatively, policies can be studied by humans at design time to develop service requestors designed to interact with specific service providers.

Recent revisions to the WS-Policy framework have extended the structure of a policy description and its associated terminology. The sections below provide a brief overview.

> NOTE
>
> The *WS-Policy language basics* section in Chapter 17 provides examples of how policies are developed using the set of languages provided by WS-Policy specifications.

7.4.2 Policy assertions and policy alternatives

The service properties expressed by a policy description are represented individually by *policy assertions*. A policy description therefore is comprised of one or more policy assertions. Examples of policy assertions include service characteristics, preferences, capabilities, requirements, and rules. Each assertion can be marked as optional or required.

Policy assertions can be grouped into *policy alternatives*. Each policy alternative represents one acceptable (or allowable) combination of policy assertions. This gives a service provider the ability to offer service requestors a choice of policies. (Each of the bullet points in our last *In Plain English* analogy, for example, would warrant a policy assertion.)

7.4.3 Policy assertion types and policy vocabularies

Policy assertions can be further categorized through *policy assertion types*. Policy assertion types associate policy assertions with specific XSD schemas. In the same manner as XML vocabularies are defined in XSD schemas, *policy vocabularies* simply represent the collection of policy types within a given policy. Similarly, a *policy alternative vocabulary* refers to the policy types contained within a specific policy alternative.

7.4.4 Policy subjects and policy scopes

A policy can be associated with a Web service, a message, or another resource. Whatever a policy is intended for is called a *policy subject*. Because a single policy can have more than one subject, the collection of a policy's subjects is referred to as the *policy scope*.

7.4.5 Policy expressions and policy attachments

Policy assertions that are physically implemented using the WS-Policy language are referred to as *policy expressions*. In other words, a policy expression is simply the XML statement used to express a policy assertion in a manner so that it can be programmatically processed. Policy expressions are physically bound to policy scopes using *policy attachments*.

7.4.6 What you really need to know

If your head is spinning at this point, don't worry. Of the many concepts we just introduced, you only need to retain the following key terms to maintain a conceptual understanding of polices:

- policy
- policy alternative
- policy assertion
- policy attachment

Let's now finish this section with a look at how policies are used by other WS-* extensions and SOA as a whole.

7.4.7 Policies in coordination

When the WS-Coordination context coordination service generates context information for participating services, it can make the distribution of context data subject to the validation of security credentials and other forms of policy information. To enforce these requirements, WS-Coordination can incorporate rules established in policies.

7.4.8 Policies in orchestration and choreography

Policies can be applied to just about any subjects that are part of orchestrations or choreographies. For example, a policy can establish various requirements for orchestration partner services and choreography participants to interact.

7.4.9 Policies in reliable messaging

The WS-ReliableMessaging specification depends on the use of the WS-Policy framework to enable some of its most fundamental features. Policies are used to implement delivery assurances through the attachment of policy assurances to the messages that take part in reliable messaging exchanges. A further set of policy assertions is provided to add various supplemental rules, constraints and reliability requirements.

7.4.10 Policies and SOA

If an SOA is a city, then policies are certainly the laws, regulations, and guidelines that exist to maintain order among inhabitants. Policies are a necessary requirement to building enterprise-level service-oriented environments, as they provide a means of communicating constraints, rules, and guidelines for just about any facet of service interaction. As a result, they improve the overall quality of the loosely coupled arrangement services are required to maintain (Figure 7.21).

Policies allow services to express so much more about themselves beyond the fundamental data format and message exchange requirements established by WSDL definitions. And policies enable services to broaden the range of available metadata while still allowing them to retain their respective independence.

The use of policies increases SOA's quality of service level by restricting valid message transmissions to those that conform to policy rules and requirements. A side benefit of inserting endpoint level constraints is that the application logic underlying services is not required to perform as much custom exception handling to deal with invalid message submissions.

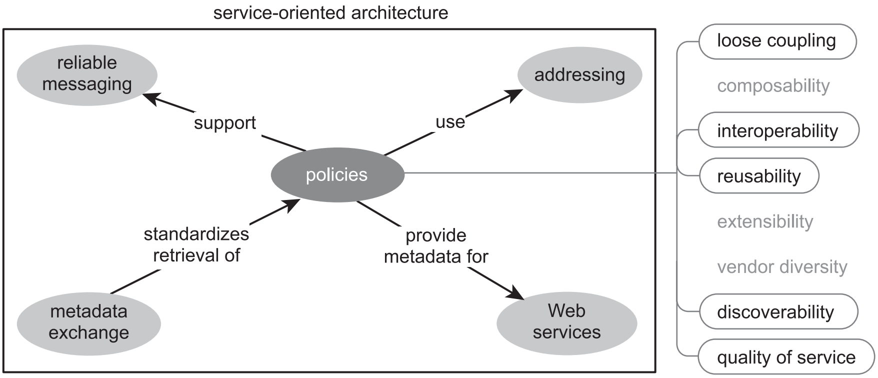

Figure 7.21
Policies relating to other parts of SOA.

Polices naturally improve the ability for services to achieve better levels of interoperation because so much more information about service endpoints can be expressed and published. Finally, because they increase the richness of service contracts, they open the door to dynamic discovery and binding.

CASE STUDY

TLS recently upgraded some of its middleware, which now provides support for the most recent version of the WS-ReliableMessaging specification. TLS wants to utilize this support but realizes that its partners still may need to continue using the previous version of WS-ReliableMessaging for some time. As a result, it chooses to support both versions by issuing a policy document containing a policy alternative.

This policy alternative states that its Vendor Profile Service will accept invoice submission sequence headers that conform to both versions of WS-ReliableMessaging, but it also expresses the fact that the newer version is preferred by TLS.

Later, TLS expands these policy assertions to include a requirement for a specific message encoding type. Regardless of which alternative is chosen by a service requestor, the same text encoding format is required.

SUMMARY OF KEY POINTS

- The WS-Policy framework provides a means of attaching properties (such as rules, behaviors, requirements, and preferences) to Web resources, most notably Web services.
- Individual properties are represented by policy assertions, which can be marked as optional or required. This allows a service to communicate non-negotiable and preferred policies.
- WS-Policy can be incorporated within the majority of WS-* extensions.
- Polices add an important layer of metadata to SOAs that increases the interoperability and discovery potential for services, while also elevating the overall quality of messaging within SOA.

7.5 Metadata exchange

When we first introduced the concept of loose coupling in Chapter 3, we explained that the sole requirement for a service requestor to interact with a service provider acting as the ultimate receiver is that the service requestor be in possession of the service provider's service description. The WSDL definition, along with any associated XSD schemas, provides the basic set of metadata required to send valid SOAP messages for consumption by the service provider.

Having just covered policies in the previous section, it is clear that, when used, policies add another important layer to the metadata stack. Using policies, our service requestor now can send SOAP messages that comply with both the WSDL interface requirements and the associated policy assertions.

Again, though, regardless of how much metadata a service makes available, the fact is that we still need to retrieve this information by either:

- manually locating it by searching for published documents
- manually requesting it by contacting the service provider entity (the service owner)
- programmatically retrieving it via a public service registry
- programmatically retrieving it by interacting with proprietary interfaces made available by the service provider entity

With the exception of using the public service registry, none of these options are particularly attractive or efficient. It would be ideal if we could simply send a standardized request such as, "give me all of the information I need to evaluate and interact with your service provider." This is exactly what *metadata exchange* accomplishes (Figure 7.22).

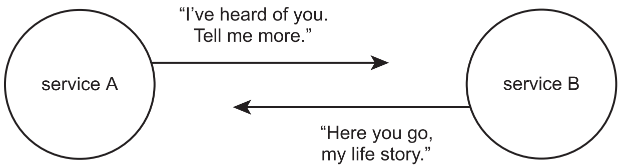

Figure 7.22
Metadata exchanges let service requestors ask what they want to know about service providers.

IN PLAIN ENGLISH

As the workload at our car wash increases, we get to the point where we are ready to hire a new worker on a full-time basis. Instead of posting an advertisement, we decide to approach a number of people we already know.

Our first request of interested candidates is that they provide us with a résumé. Because we want to check references, we always look through the résumé to see if references are attached. Sometimes they are, but most of the time it simply states that references are available upon a separate request. As a result, we contact the candidate again to request the references document.

This analogy demonstrates the simplicity of the metadata exchange concept. We first issue a request from a resource for (meta) information about that resource. If the information we receive is not sufficiently complete, we issue a second request for the remaining (meta) information.

7.5.1 The WS-MetadataExchange specification

This specification essentially allows for a service requestor to issue a standardized request message that asks for some or all of the meta information relating to a specific endpoint address.

In other words, if a service requestor knows where a service provider is located, it can use metadata exchange to request all service description documents that comprise the service contract by sending a metadata exchange request to the service provider.

Originally the WS-MetadataExchange specification specified the following three types of request messages:

- Get WSDL
- Get Schema
- Get Policy

Even though these represent the three most common types of meta information currently attached to Web services, the specification authors realized that future metadata documents would likely emerge. A subsequent revision therefore resulted in a single type of request message:

- Get Metadata

This message is further supplemented by the Get request message. Both are explained in the following sections.

NOTE
To see examples of WS-MetadataExchange request and response messages, see the *WS-MetadataExchange language basics* section in Chapter 17.

7.5.2 Get Metadata request and response messages

As previously mentioned, a service requestor can use metadata exchange to programmatically request available metadata documents associated with a Web service. To do so, it must issue a *Get Metadata request* message. This kicks off a standardized request and response MEP resulting in the delivery of a *Get Metadata response* message.

Here's what happens for a metadata retrieval activity to successfully complete:

1. A service requestor issues the Get Metadata request message. This message can request a specific type of service description document (WSDL, XSD schema, policy), or it can simply request that all available metadata be delivered.
2. The Get Metadata request message is received at the endpoint to which it is delivered. The requested meta information is documented in a Get Metadata response message.
3. The Get Metadata response message is delivered to the service requestor. The contents of this message can consist of the actual metadata documents, address references to the documents, or a combination of both.

Figure 7.23 illustrates these steps.

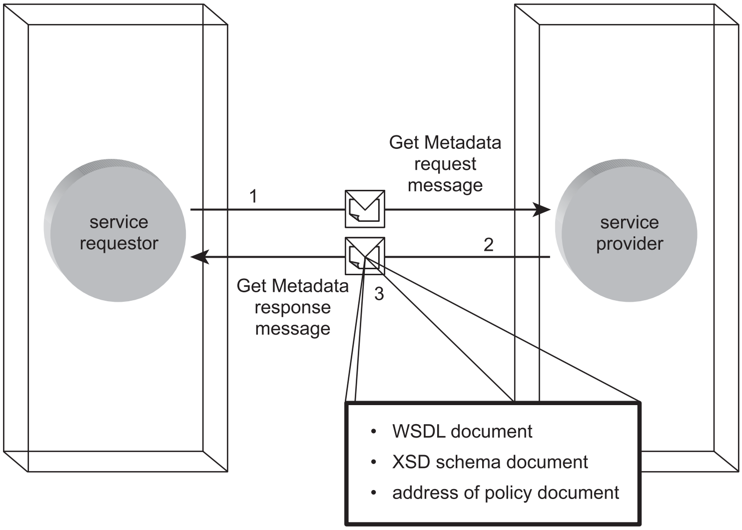

Figure 7.23
Contents of a sample Get Metadata response message.

7.5.3 Get request and response messages

In Step 3 of the preceding scenario, we explained how the Get Metadata response message does not need to actually contain all of the requested metadata. It can simply provide a list of URIs that point to the separate documents.

To allow the retrieval of *all* meta information to be fully automated, the WS-MetadataExchange specification provides a means for the service requestor to explicitly request the document content for any references that were provided as part of the original Get Metadata response message. It achieves this through the use of the *Get request* and *Get response* messages.

Here's a brief description of the steps involved in this sub-process:

1. Upon receiving the Get Metadata response message, the service requestor determines that it would like to receive the actual content of the metadata documents for which it only received references. As a result, the service requestor issues a Get request message indicating which metadata information it would like retrieved.
2. The Get request message is received at the endpoint to which it was delivered. The requested data is placed into a Get response message.

3. The Get response message is delivered to the service requestor.

Figure 7.24 shows the execution sequence of these steps, which should provide the service requestor with all the information it needs (and therefore concludes the metadata exchange process).

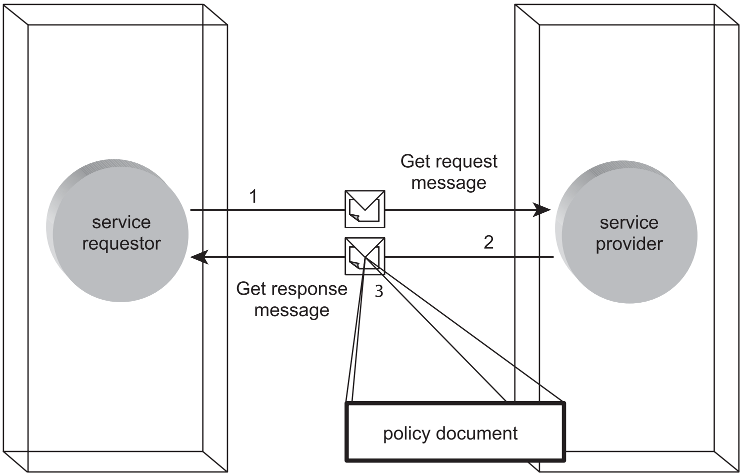

Figure 7.24
Contents of a sample Get response message.

7.5.4 Selective retrieval of metadata

Meta documents describing services with comprehensive interfaces and features can be large in size, especially when assembled into one mega-description. Use of the selective Get request message type therefore reduces the chances of unnecessary information being transported.

The Get Metadata response message first sends along what is considered the essential piece of service meta information. It is then up to the service requestor to determine what further metadata it requires. (Note that the endpoint to which a Get Metadata request message is sent can represent multiple WSDL, XSD schema, and policy documents.)

7.5.5 Metadata exchange and service description discovery

It also is important to note that metadata exchange does not really help service requestors discover service providers. Service registries, such as those implemented using the UDDI standard, can be used to discover service descriptions that meet certain search criteria. While service registries also provide location information for the actual WSDL definition of a service, they can be used in conjunction with metadata exchange messages.

Essentially, a service requestor could first query a public registry to retrieve the endpoint addresses of any Web service candidates that appear to provide the sought-after features. The same requestor could then employ metadata exchange to contact each candidate and request associated metadata documents. This would give the service requestor more information to better assess which service provider it should be working with. It also would provide the service requestor with all of the details it needs to begin interacting with the chosen service. So while it may not further the cause of attaining discoverable services, it does support discovery by rounding out the overall dynamic discovery process.

7.5.6 Metadata exchange and version control

So far we've focused on the ability of metadata exchange to enable service requestors to retrieve any necessary meta information for them to begin interacting with service providers. Another important aspect of this WS-* extension is its potential to automate the administration of service contracts.

As services evolve, the nature and scope of the functionality they deliver can undergo alterations. This can result in changes trickling down to the service meta layer, which, in turn, can lead to new versions of a service's WSDL, XSD schema, or policy documents.

This raises the age-old version control challenges. Service requestors already interacting with a service provider either need to be notified ahead of time of upcoming changes, or they need to be supported with an outdated service description.

Some services-based solutions have dealt with this problem by building custom operations that can be used to retrieve the latest service description (metadata) information. While the same functionality is essentially provided by metadata exchange, the main benefit of its use is that it implements this feature in a standardized manner. Now any service-oriented application that supports metadata exchange can allow service requestors to retrieve the latest service contract as often as they like.

When changes to meta information are expected to occur frequently, a service requestor could be programmed to periodically retrieve available metadata documents to compare them to the documents already in use. In fact, service requestors could even build metadata exchange features into their exception handling. If a standard SOAP request is rejected by the service provider as a result of an interface, schema, or policy incompatibility error, the service requestor's exception handling routine could respond by retrieving and checking the latest metadata documents.

7.5.7 Metadata exchange and SOA

The simple concepts behind metadata exchange support some key aspects of SOA (Figure 7.25). Its ability to automate the retrieval of meta information reinforces loose coupling between services, and increases the ability for service requestors to learn more about available service providers. By standardizing access to and retrieval of metadata, service requestors can programmatically query a multitude of candidate providers. Because enough service provider metadata can more easily be retrieved and evaluated, the overall discovery process is improved, and the likelihood for services to be reused is increased.

By establishing a standardized means of service description exchange, this extension can vastly improve interoperability when broadly applied to volatile environments. By being able to query service providers prior to attempting access, requestors can verify that the correct metadata is in fact being used for their planned message exchanges. This can increase the QoS factor of SOA, as it tends to avoid a multitude of maintenance problems associated with service contract changes.

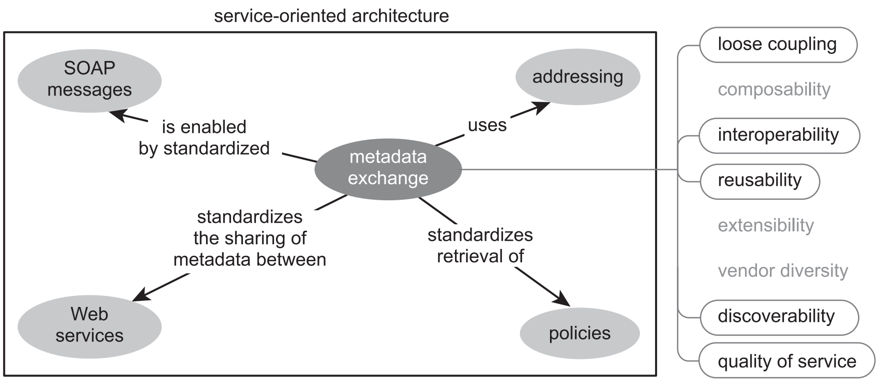

Figure 7.25
Metadata exchange relating to other parts of SOA.

It is also worth mentioning that metadata exchange reduces the need for developers to attain meta information at design time and eliminates the need for custom-developed metadata retrieval extensions. Finally, the dynamic exchange of service descriptions can lead to the potential of automating version control and other metadata-related functions.

CASE STUDY

As TLS continues to evolve its B2B solution, new features are added and some existing functionality is modified. This can, occasionally, result in changes to the WSDL interface definitions of TLS services, as well as revisions to service policies. Any of these changes can obviously affect the online partners that regularly connect to TLS.

Therefore, all public TLS services support the processing of WS-MetadataExchange requests. At the onset, partners who register for the TLS B2B solution are strongly encouraged to issue Get Metadata request messages frequently to receive the latest service contracts.

RailCo learned about this the hard way. To date they never bothered incorporating metadata exchange functionality within their services, as they were not required to do so. After a change to the TLS Accounts Payable Service WSDL, though, the RailCo Invoice Submission Service submitted an invoice message that was rejected by TLS.

The resulting error description was unclear, and exception handling logic within the RailCo service assumed this condition was the result of the TLS service being unavailable. It was therefore designed to periodically retry the message submission on a daily basis. Only after three days did someone at RailCo notice that an acknowledgement had not been received from TLS. A lengthy investigation led to the eventual discovery that the failed submissions were the result of a change to the TLS WSDL definition.

As a result of this experience, RailCo revised their Invoice Submission Service to interact with the metadata exchange functionality offered by TLS (Figure 7.26). The service now issues a periodic Get Metadata message to the TLS Accounts Payable Service.

The Accounts Payable Service responds with a Get Metadata response message containing its current WSDL, XSD schema, and policy information. The RailCo Invoice Submission Service verifies that the service description documents used by RailCo match those currently published by the TLS service.

If the verification succeeds, it's business as usual, and RailCo proceeds to issue invoice submission messages. If the metadata does not match, a special error condition is raised at RailCo's end, and no further invoices are sent until it is addressed by an administrator.

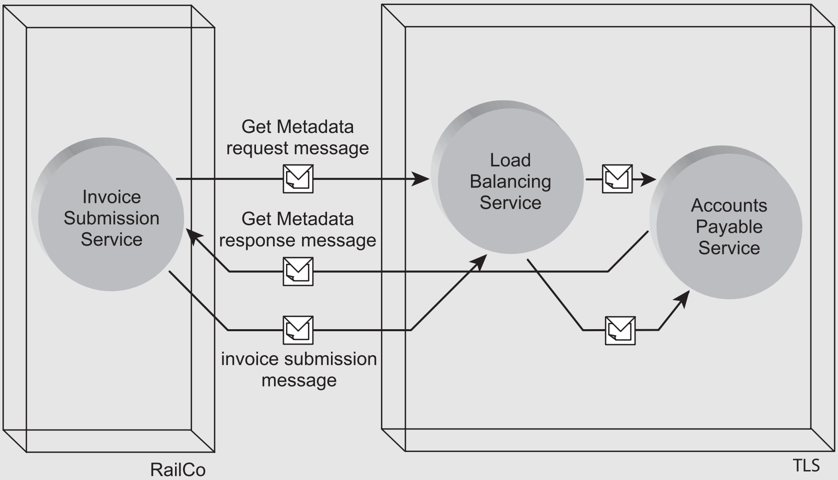

Figure 7.26
The revised RailCo Invoice Submission Process now includes a periodic metadata exchange with TLS.

SUMMARY OF KEY POINTS

- Metadata exchange allows service requestors to issue request messages to retrieve metadata for service providers.
- The WS-MetadataExchange specification standardizes two types of request messages: the Get Metadata request (which returns metadata content and/or references) and the Get request (which returns the content of a previously returned reference).
- Metadata exchange assists in improving the service description discovery process and in alleviating version control issues related to service meta information.
- Automated metadata retrieval leads to several standardized improvements within SOA and reinforces the loosely coupled nature of autonomous services.

7.6 Security

Security requirements for automation solutions are nothing new to the world of IT. Similarly, service-oriented applications need to be outfitted to handle many of the traditional security demands of protecting information and ensuring that access to logic is only granted to those permitted.

However, the SOAP messaging communications framework, upon which contemporary SOA is built, emphasizes particular aspects of security that need to be accommodated by a security framework designed specifically for Web services.

A family of security extensions parented by the WS-Security specification comprise such a framework, further broadened by a series of supplementary specifications with specialized feature sets. Sidebar 7.1 provides a list of current security-related specifications. While we clearly cannot discuss concepts for all of them, it is worth spending some time looking at the basic functions performed by the following three core specifications:

- WS-Security
- XML-Signature
- XML-Encryption

Additionally, we'll briefly explore the fundamental concepts behind *single sign-on*, a form of centralized security that complements these WS-Security extensions.

Sidebar 7.1 A list of security specifications that may be used as part of SOA. For more information regarding these specifications, visit: `www.specifications.ws`.

- WS-Security
- WS-SecurityPolicy
- WS-Trust
- WS-SecureConversation
- WS-Federation
- Extensible Access Control Markup Language (XACML)
- Extensible Rights Markup Language (XrML)
- XML Key Management (XKMS)
- XML-Signature
- XML-Encryption
- Security Assertion Markup Language (SAML)
- .NET Passport
- Secure Sockets Layer (SSL)
- WS-I Basic Security Profile

Before we begin, it is worth noting that this section organizes security concepts as they pertain to and support the following five common security requirements: identification, authentication, authorization, confidentiality, and integrity.

NOTE

For an overview of the core language elements from the WS-Security, XML-Encryption, and XML-Signature languages, see the *WS-Security language basics* section in Chapter 17.

IN PLAIN ENGLISH

Toward the end of a working day, Jim leaves the car wash early. He has an appointment with someone selling a used power washer that we are interested in buying. Before he can meet this person, Jim must stop by the bank to withdraw a fair amount of money for the potential purchase (the seller has stated that this must be a cash sale). I also ask Jim to do me a favor and pick up a package that's waiting for me at a postal outlet near the bank.

Jim agrees and proceeds on his errand trip. Upon entering the bank, Jim must fill out a withdrawal slip on which he is asked to identify himself by writing his full name. Jim then comes face-to-face with a bank teller who, upon seeing that he wants to make a withdrawal, requests that he produce a bank card and one piece of photo ID.

Jim shows the teller his business account card and his driver's license, which the teller subsequently verifies. After it is confirmed that Jim is who he stated he was on the withdrawal slip, the teller asks Jim to enter his bank card pass code. This further ensures that he is an individual allowed to make this type of withdrawal.

With the money in hand, Jim proceeds to the postal outlet. There he presents the notification card I received in the mail indicating that a parcel is being held for me. Jim states his name (and therefore does not claim to be the same person whose name is on the notification card) and also states that he is here to pick up the parcel for someone else. The employee at the postal outlet asks Jim for ID, so he pulls out his driver's license again. Upon reviewing the information on the driver's license and the notification card, the employee informs Jim that he cannot pick up this package.

Jim's experience at the bank required that he go through three levels of clearance: identification (withdrawal slip), authentication (bank card and photo ID), and authorization (pass code and bank record). While no security was really applied to the identification part of this process, it did kick off the remaining two security phases for which Jim satisfied requirements (and for which reason he subsequently received the requested money).

At the post office, though, Jim did not pass the authorization stage. Only individuals that share the last name or reside at the same address of the person identified on the notification card are allowed to pick up deliveries on their behalf. Jim's claimed identity was authenticated by the driver's license, but because Jim is not a relative of mine and does not live at the same address as I do, he did not meet the requirement that would have authorized him to pick up the parcel.

7.6.1 Identification, authentication, and authorization

For a service requestor to access a secured service provider, it must first provide information that expresses its origin or owner. This is referred to as making a *claim* (Figure 7.27). Claims are represented by identification information stored in the SOAP header. WS-Security establishes a standardized header block that stores this information, at which point it is referred to as a *token*.

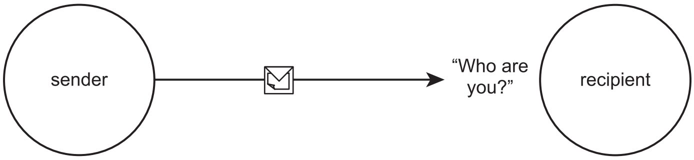

Figure 7.27
An identity is a claim made regarding the origin of a message.

Authentication requires that a message being delivered to a recipient prove that the message is in fact from the sender that it claims to be (Figure 7.28). In other words, the service must provide proof that its claimed identity is true.

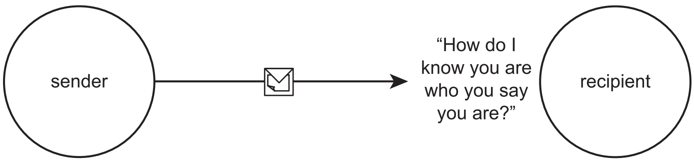

Figure 7.28
Authentication means proving an identity.

Once authenticated, the recipient of a message may need to determine what the requestor is allowed to do. This is called *authorization* (Figure 7.29).

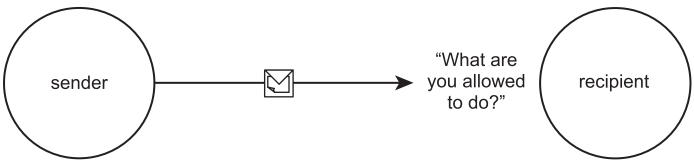

Figure 7.29
Authorization means determining to what extent authentication applies.

7.6.2 Single sign-on

A challenge facing the enablement of authentication and authorization within SOA is propagating the authentication and authorization information for a service requestor across multiple services behind the initial service provider. Because services are autonomous and independent from each other, a mechanism is required to persist the security context established after a requestor has been authenticated. Otherwise, the requestor would need to re-authenticate itself with every subsequent request.

The concept of single sign-on addresses this issue. The use of a single sign-on technology allows a service requestor to be authenticated once and then have its security context information shared with other services that the requestor may then access without further authentication.

There are three primary extensions that support the implementation of the single sign-on concept:

- SAML (Security Assertion Markup Language)
- .NET Passport
- XACML (XML Access Control Markup Language)

As an example of a single sign-on technology that supports centralized authentication and authorization, let's briefly discuss some fundamental concepts provided by SAML.

SAML implements a single sign-on system in which the point of contact for a service requestor can also act as an *issuing authority*. This permits the underlying logic of that service not only to authenticate and authorize the service requestor, but also to assure the other services that the service requestor has attained this level of clearance.

Other services that the service requestor contacts, therefore, do not need to perform authentication and authorization steps. Instead, upon receiving a request, they simply contact the issuing authority to ask for the authentication and authorization clearance it originally obtained. The issuing authority provides this information in the form of *assertions* that communicate the security details. (The two types of assertions that contain authentication and authorization information are simply called *authentication assertions* and *authorization assertions*.)

In Figure 7.30 we illustrate some of the mechanics behind SAML.

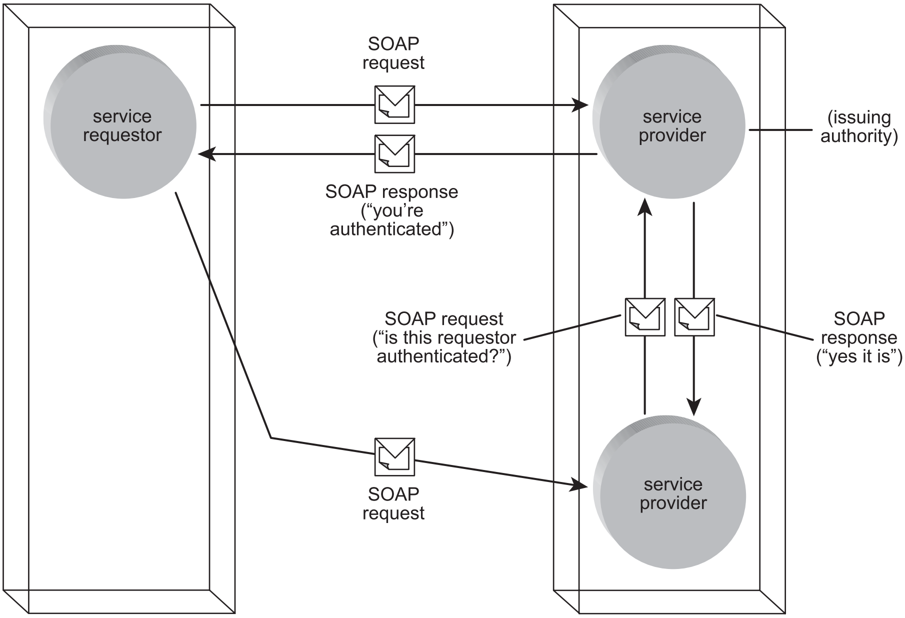

Figure 7.30
A basic message exchange demonstrating single sign-on (in this case, as implemented by SAML).

7.6.3 Confidentiality and integrity

Confidentiality is concerned with protecting the privacy of the message contents (Figure 7.31). A message is considered to have remained confidential if no service or agent in its message path not authorized to do so viewed its contents.

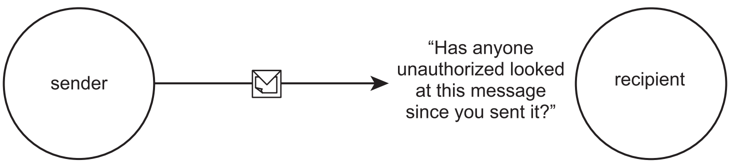

Figure 7.31
Confidentiality means that the privacy of the message has been protected throughout its message path.

Integrity ensures that a message has not been altered since its departure from the original sender (Figure 7.32). This guarantees that the state of the message contents remained intact from the time of transmission to the point of delivery.

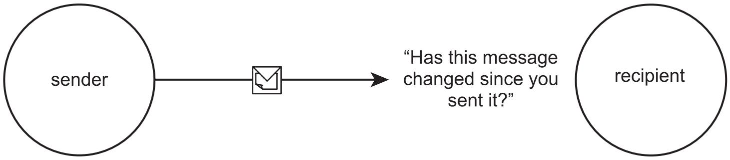

Figure 7.32
Integrity means ensuring that a message's contents have not changed during transmission.

7.6.4 Transport-level security and message-level security

The type of technology used to protect a message determines the extent to which the message remains protected while making its way through its message path. Secure Sockets Layer (SSL), for example, is a very popular means of securing the HTTP channel upon which requests and responses are transmitted. However, within a Web services-based communications framework, it can only protect a message during the transmission between service endpoints. Hence, SSL only affords us *transport-level* security (Figure 7.33).

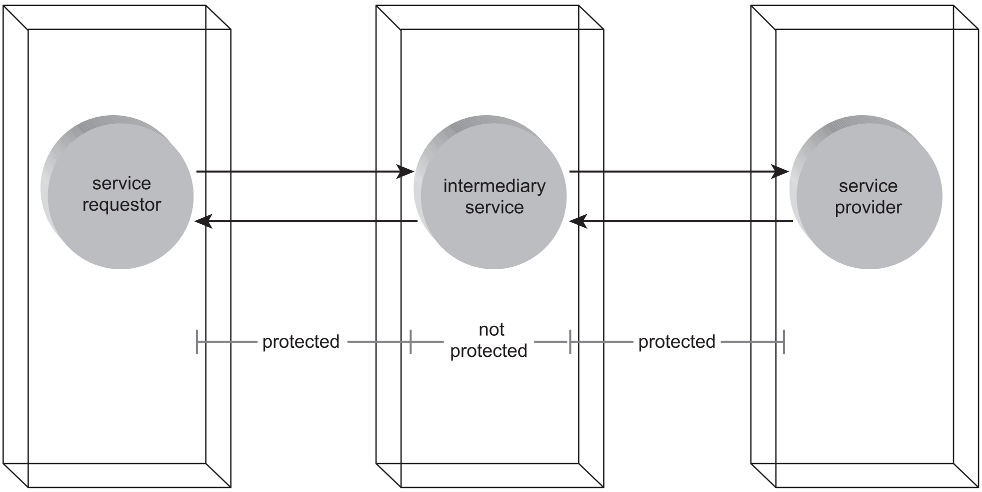

Figure 7.33
Transport-level security only protects the message during transit between service endpoints.

If, for example, a service intermediary takes possession of a message, it still may have the ability to alter its contents. To ensure that a message is fully protected along its entire message path, *message-level security* is required (Figure 7.34). In this case, security measures are applied to the message itself (not to the transport channel on which the message travels). Now, regardless of where the message may travel, the security measures applied go with it.

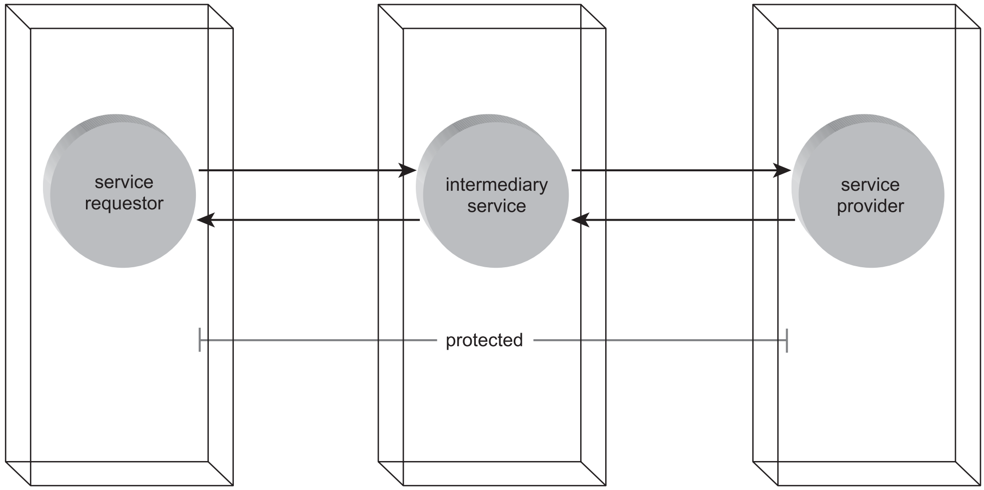

Figure 7.34
Message-level security guarantees end-to-end message protection.

7.6.5 Encryption and digital signatures

Message-level confidentiality for an XML-based messaging format, such as SOAP, can be realized through the use of specifications that comprise the WS-Security framework. In this section we focus on XML-Encryption and XML-Signature, two of the more important WS-Security extensions that provide security controls that ensure the confidentiality and integrity of a message.

XML-Encryption, an encryption technology designed for use with XML, is a cornerstone part of the WS-Security framework. It provides features with which encryption can be applied to an entire message or only to specific parts of the message (such as the password).

To ensure message integrity, a technology is required that is capable of verifying that the message received by a service is authentic in that it has not been altered in any manner since it first was sent. XML-Signature provides features that allow for an XML document to be accompanied by a special algorithm-driven piece of information that represents a digital signature. This signature is tied to the content of the document so that verification of the signature by the receiving service only will succeed if the content has remained unaltered since it first was sent.

> **NOTE**
>
> Digital signatures also support the concept of non-repudiation, which can prove that a message containing a (usually legally binding) document was sent by a specific requestor and delivered to a specific provider.

As illustrated in Figure 7.35, XML-Encryption can be applied to parts of a SOAP header, as well as the contents of the SOAP body. When signing a document, the XML-Signature can reside in the SOAP header.

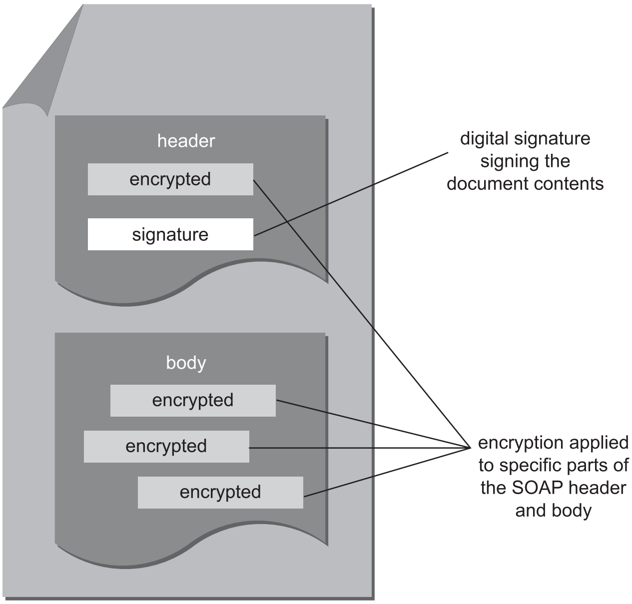

Figure 7.35
A digitally signed SOAP message containing encrypted data.

> **NOTE**
>
> Both encryption and digital signature technologies rely on the use of *keys*. These are special values used to unlock the algorithm upon which encryption and digital signatures are based. *Shared keys* are typically used by encryption technologies and require that both the sender and receiver of a message use the same key. *Public/private key pairs* are commonly used by digital signature technologies, where the message sender signs the document with a key that is different from the one used by the recipient. (One of the keys is public, but the other is private.)

7.6.6 Security and SOA

Message-level security can clearly become a core component of service-oriented solutions. Security measures can be layered over any message transmissions to either protect the message content or the message recipient. The WS-Security framework and its accompanying specifications therefore fulfill fundamental QoS requirements that enable enterprises to:

- utilize service-oriented solutions for the processing of sensitive and private data
- restrict service access as required

As shown in Figure 7.36, the security framework provided by WS-Security also makes use of the WS-Policy framework explained earlier (a separate specification called WS-SecurityPolicy provides a series of supporting policy assertions).

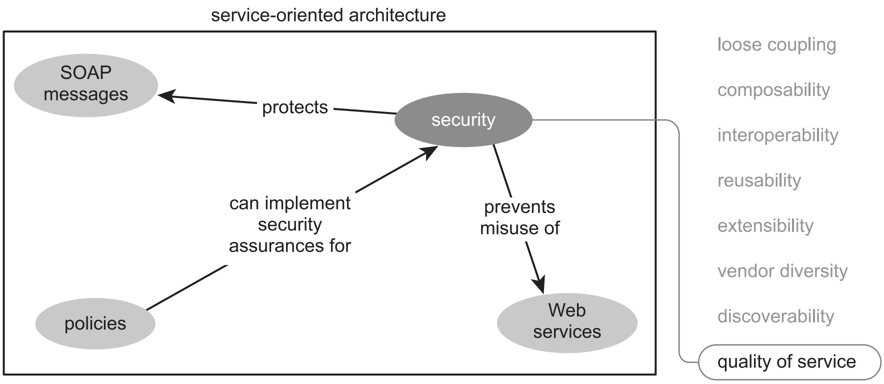

Figure 7.36
Security, as it relates to policies, SOAP messages, and Web services.

CASE STUDY

TLS has a message-level security policy that applies to any business documents sent to its B2B solution.

The policy has the following rules:

- Any dollar values residing in documents sent via SOAP messages must be encrypted.
- Any invoice submitted to TLS with a total dollar value of over $30,000 must also be digitally signed.

To comply with this policy, RailCo is required to apply XML-Encryption to the parts of the invoice message sent by the Invoice Submission Service that contain monetary values.

It further embeds a business rule into the Invoice Submission Service's underlying logic that checks for invoice totals that exceed the $30,000 mark. Those that do, have their corresponding SOAP message documents digitally signed using XML-Signature.

SUMMARY OF KEY POINTS

- Security within SOA is a multi-faceted subject matter that encompasses the feature set of numerous specifications. The WS-Security framework governs a subset of these specifications, and establishes a cohesive and composable security architecture.
- The primary aspects of security addressed by these specifications are identification, authentication, authorization, integrity, and confidentiality, as well as non-repudiation.
- Two primary technologies for preserving the integrity and confidentiality of XML documents are XML-Encryption and XML-Signature.

7.7 Notification and eventing

With its roots in the messaging-oriented middleware era, the publish-and-subscribe MEP introduces a composite messaging model, comprised of primitive MEPs that implement a push delivery pattern. It establishes a unique relationship between service providers and service requestors where information is exchanged (often blindly) to achieve a form of dynamic *notification* (Figure 7.37).

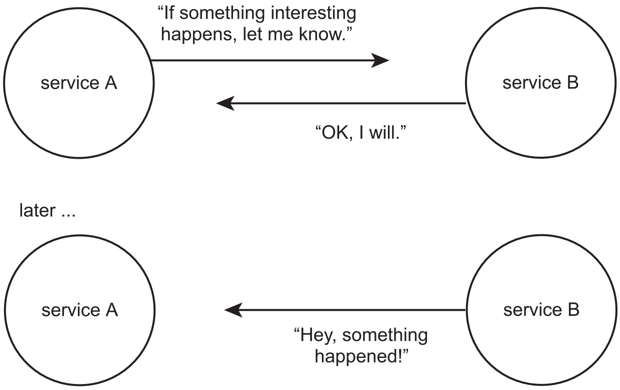

Figure 7.37
Once subscribed, service A is notified of anything service B publishes that is of interest to service A.

While notification itself can be applied to different types of MEPs, the focus of this section is a discussion of notification within the context of the publish-and-subscribe pattern.

7.7.1 Publish-and-subscribe in abstract

As explained in Chapter 6, this messaging pattern can be classified as a complex MEP assembled from a series of primitive MEPs. It involves a *publisher* service that makes information categorized by different *topics* available to registered *subscriber* services. Subscribers can choose which topics they want to register for, either by interacting with the publisher directly or by communicating with a separate *broker* service. A topic is an item of interest and often is tied to the occurrence of an event.

When a new piece of information on a given topic becomes available, a publisher broadcasts this information to all those services that have subscribed to that topic. Alternatively, a *broker* service can be used to perform the broadcast on the publisher's behalf. This decouples the publisher from the subscriber, allowing each to act independently and without knowledge of each other.

IN PLAIN ENGLISH

Both our car wash company and our partner's are members of the World-Wide Car Washing Consortium (W3CC), an international organization dedicated to the advancement of the field of car washing. This organization issues weekly bulletins on a number of different topics. Members can sign up for the bulletins that are of most interest to them.

Our partner wants to stay informed with most of what occurs in the car washing industry, so they are registered to receive almost all of the bulletins. We are more interested in advancements relating to soap technology and sponging techniques. Our company, therefore, only subscribes to bulletins that discuss these topics.

Whenever industry developments (events) occur that we have expressed an interest in and for as long as our subscriptions are valid, bulletins (notifications) are sent to us (the subscribers).

7.7.2 One concept, two specifications

Two major implementations of the publish-and-subscribe pattern exist:

- The WS-Notification framework
- The WS-Eventing specification

Spearheaded by IBM and Microsoft respectively, these use different approaches and terminology to cover much of the same ground. It is expected that a single publish-and-subscribe specification eventually will emerge as an industry standard. The remainder of this section is dedicated to exploring features of both specifications.

7.7.3 The WS-Notification Framework

As with other WS-* frameworks, what is represented by WS-Notification is a family of related extensions that have been designed with composability in mind.

- WS-BaseNotification—Establishes the standardized interfaces used by services involved on either end of a notification exchange.
- WS-Topics—Governs the structuring and categorization of topics.
- WS-BrokeredNotification—Standardizes the broker intermediary used to send and receive messages on behalf of publishers and subscribers.

> **NOTE**
>
> To improve clarity, we take the liberty of breaking up some of the large concatenated terms provided in the WS-Notification specifications. For example, the term "NotificationMessage" in the WS-BaseNotification specification is expressed as "notification message" in this section.

Situations, notification messages, and topics

The notification process typically is tied to an event that is reported on by the publisher. This event is referred to as a *situation*. Situations can result in the generation of one or more *notification messages*. These messages contain information about (or relating to) the situation, and are categorized according to an available set of *topics*. Through this categorization, notification messages can be delivered to services that have subscribed to corresponding topics.

Notification producers and publishers

So far we've been using the familiar "publisher" and "subscriber" terms to describe the roles services assume when they participate in the publish-and-subscribe pattern. Within WS-Notification, however, these terms have more distinct definitions.

The term *publisher* represents the part of the solution that responds to situations and is responsible for generating notification messages. However, a publisher is not necessarily required to distribute these messages. Distribution of notification messages is the task of the *notification producer*. This service keeps track of subscriptions and corresponds directly with subscribers. It ensures that notification messages are organized by topic and delivered accordingly.

Note that:

- A publisher may or may not be a Web service, whereas the notification producer is always a Web service.
- A single Web service can assume both publisher and notification producer roles.
- The notification producer is considered the service provider.

Notification consumers and subscribers

A *subscriber* is the part of the application that submits the subscribe request message to the notification producer. This means that the subscriber is not necessarily the recipient of the notification messages transmitted by the notification producer. The recipient is the *notification consumer*, the service to which the notification messages are delivered (Figure 7.38).

Note that:

- A subscriber does not need to exist as a Web service, but the notification consumer is a Web service.
- Both the subscriber and notification consumer roles can be assumed by a single Web service.
- The subscriber is considered the service requestor.

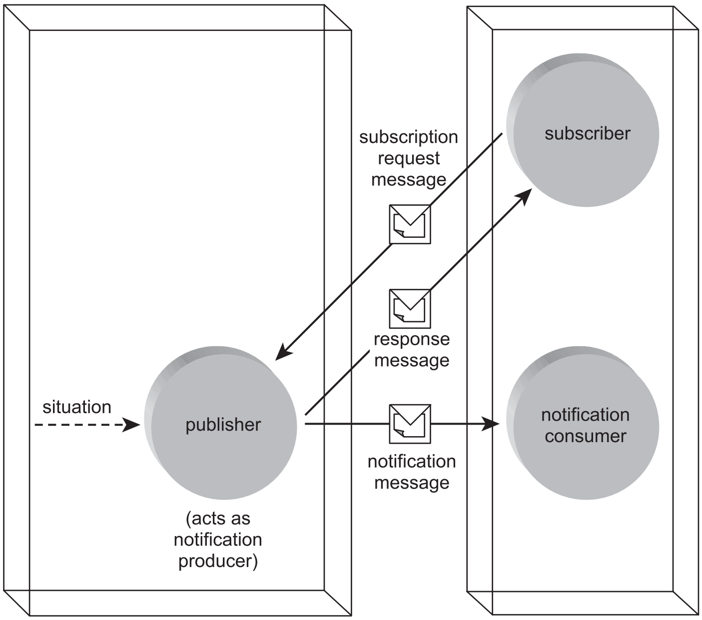

Figure 7.38
A basic notification architecture.

Notification broker, publisher registration manager, and subscription manager
To alleviate the need for direct contact between the two groups of services we described in the previous two sections, a set of supplementary services is available (Figure 7.39).

- The *notification broker*—A Web service that acts on behalf of the publisher to perform the role of the notification producer. This isolates the publisher from any contact with subscribers. Note that when a notification broker receives notification messages from the publisher, it temporarily assumes the role of notification consumer.

- The *publisher registration manager*—A Web service that provides an interface for subscribers to search through and locate items available for registration. This role may be assumed by the notification broker, or it may be implemented as a separate service to establish a further layer of abstraction.
- The *subscription manager*—A Web service that allows notification producers to access and retrieve required subscriber information for a given notification message broadcast. This role also can be assumed by either the notification producer or a dedicated service.

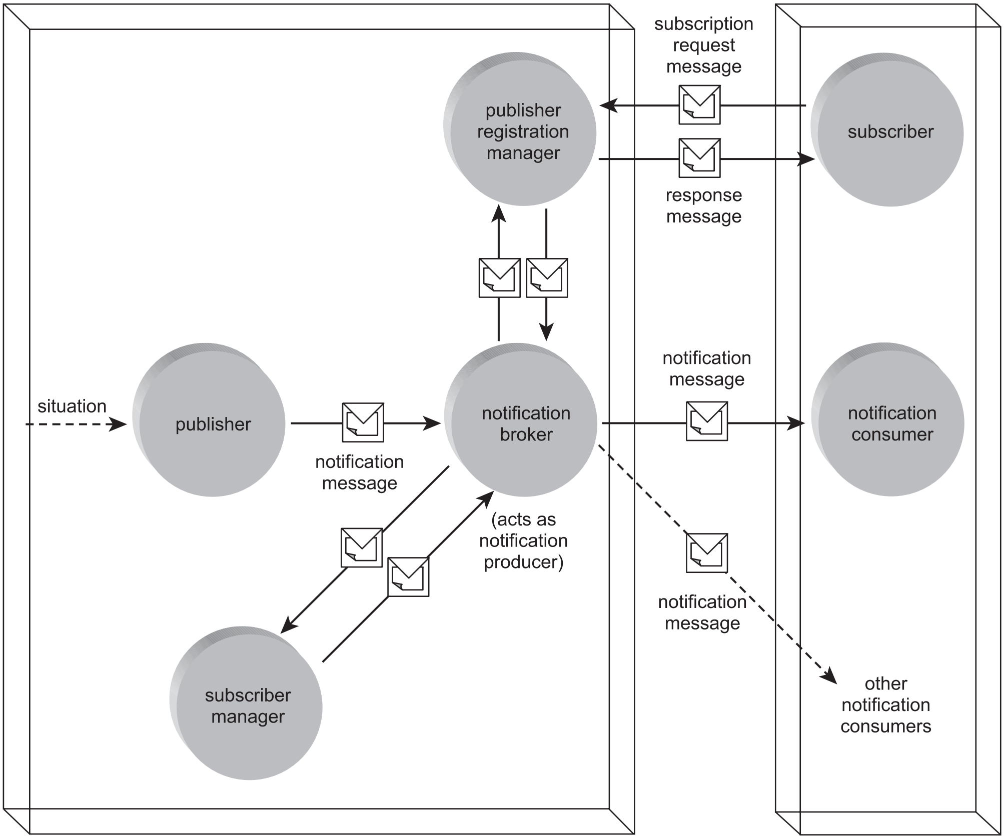

Figure 7.39
A notification architecture including a middle tier.

7.7.4 The WS-Eventing specification

As its name implies, WS-Eventing addresses publish-and-subscribe requirements by focusing on an event-oriented messaging model. When an event related to one Web service occurs, any other services that have expressed interest in the event are subsequently notified. Following are brief explanations of the terms and concepts expressed by the WS-Eventing specification.

Event sources

The term "publisher" is never actually mentioned in the WS-Eventing specification. Instead, its role is assumed by a broader-scoped Web service, known as the *event source*. This part of the eventing architecture is responsible for both receiving subscription requests and for issuing corresponding notification messages that report information about occurred events.

Event sinks and subscribers

On the subscription end of the eventing model, separate Web services manage the processing of notification and subscription messages. An *event sink* is a service designed to consume (receive) notification messages from the event source. *Subscribers* are services capable of issuing various types of subscription requests.

Subscription managers

An event source, by default, assumes the responsibility of managing subscriptions and transmitting notifications. In high volume environments it may be desirable to split these roles into separate services. To alleviate the demands on the event source, intermediate services, known as *subscription managers*, optionally can be used to distribute publisher-side processing duties.

Notification messages and subscription end messages

When an event occurs, it is reported by the event source via the issuance of a *notification message* (also called an *event message*). These are standard SOAP messages that contain WS-Eventing-compliant headers to convey event details.

WS-Eventing allows for an expiry date to be attached to subscriptions. This requires that subscribers issue renewal requests for the subscription to continue (as discussed in the next section). If a subscription is left to expire, though, it is the event source that often is expected to send a special type of notification to the corresponding event sink, called a *subscription end message*.

Subscription messages and subscription filters

Subscribers issue *subscription messages* directly to the event source or to an intermediate subscription manager. Different types of subscription-related requests can be transmitted via subscription messages.

The following specific requests are supported:

- *Subscribe*—Requests that a new subscription be created. (Note that this message also contains the filter details, as well as the endpoint destination to which a subscription end message is to be delivered. Filters are described shortly.)
- *Unsubscribe*—Requests that an existing subscription be canceled.
- *Renew*—Requests that an existing subscription scheduled to expire be renewed.
- *GetStatus*—Requests that the status of a subscription be retrieved.

For a subscriber to communicate that the event sink (on behalf of whom it is submitting the subscription request) is only interested in certain types of events, it can issue a subscription message containing a *subscription filter*. If the event source does not support filtering (or if it cannot accommodate the requested filter), the subscription request is denied.

The relationships between the subscription manager, event source, subscriber, and event sink are shown in Figure 7.40.

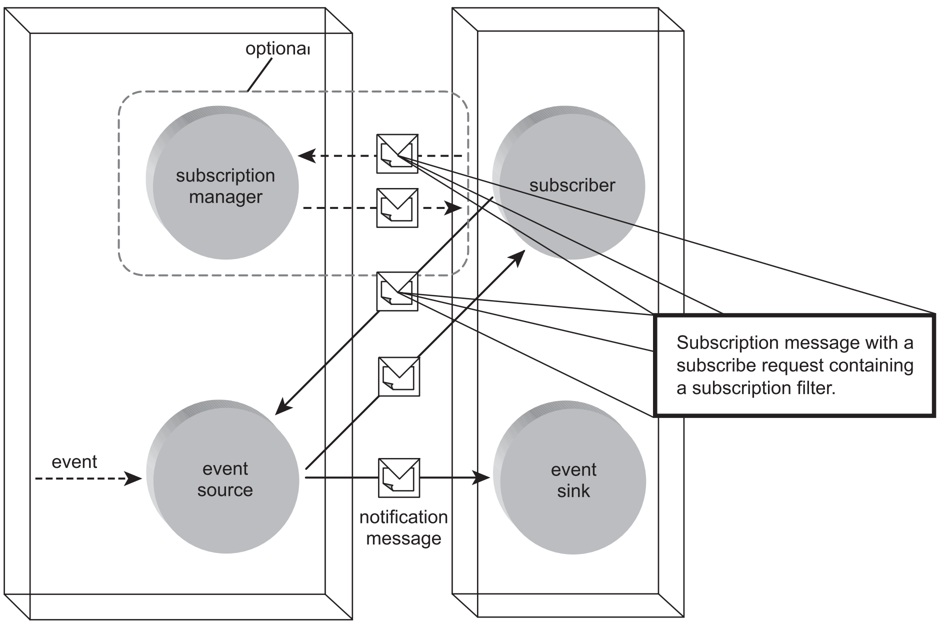

Figure 7.40
A basic eventing architecture.

7.7.5 WS-Notification and WS-Eventing

The fact that these two specifications currently provide overlapping feature sets is no indication that this will remain so in the future. It has been speculated that the reason these specifications were created separately was because the individual sponsors had diverging requirements. One of IBM's goals is to incorporate WS-Notification with its grid computing initiatives. Microsoft, on the other hand, is expected to utilize WS-Eventing within its system administration platform.

In an effort to continue promoting interoperability across proprietary platforms, IBM recently joined the WS-Eventing effort. It is entirely within the realm of possibilities that either specification will be modified to align with the other—or that the vendors involved will come to an agreement on how to establish a single notification extension that will meet their collective requirements. Language descriptions for these two specifications are therefore not currently provided in this book. (If you are interested in viewing the individual specifications, visit `www.specifications.ws`.)

7.7.6 Notification, eventing, and SOA

By implementing a messaging model capable of supporting traditional publish-and-subscribe functionality, corresponding legacy features now can be fully realized within service-oriented application environments (Figure 7.41). Moreover, the ability to weave a sophisticated notification system into service-oriented solutions can significantly broaden the applicability of this messaging model (as evidenced by the before mentioned plans to incorporate notification with grid computing).

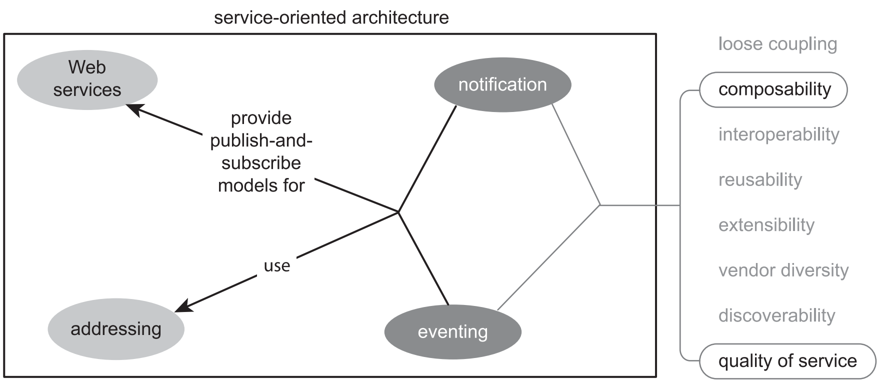

Figure 7.41

Notification and eventing establishing standardized publish-and-subscribe models within SOA.

Service-oriented solutions can increase QoS characteristics by leveraging notification mechanisms to perform various types of event reporting. For example, performance and exception management related events can trigger notification broadcasts to potential service requestors (subscribers), informing them of a variety of conditions.

CASE STUDY

In response to a series of complaints from vendors who experienced message transmission problems that resulted from changes to TLS service descriptions, TLS has decided to supplement their existing metadata exchange support by implementing a notification system. Now, business partners will be forewarned of any upcoming changes that might impact their systems.

There are many services that comprise the TLS B2B solution. Each performs a specific function that involves one or more types of partners. Not all partners need to interact with every TLS service. As a result, the notification system is set up in such a manner that partners are able to subscribe to notifications relating to specific TLS services or groups of services.

For this, TLS has provided a dedicated System Notification Service that acts as the publisher of notification messages. Partners are consequently required to implement their own subscriber services. Each notification message essentially requests that the recipient initiate a WS-MetadataExchange against the provided TLS endpoint(s).

RailCo creates a separate subscription service to interact with the TLS System Notification Service. Unfortunately, RailCo calls its new service the "TLS Subscription Service," which is sure to lead to confusion in the future. Regardless, RailCo uses its service to subscribe to and receive notifications relating to the two primary services with which it interacts on a regular basis: the TLS Accounts Payable and Purchase Order Services (Figure 7.42).

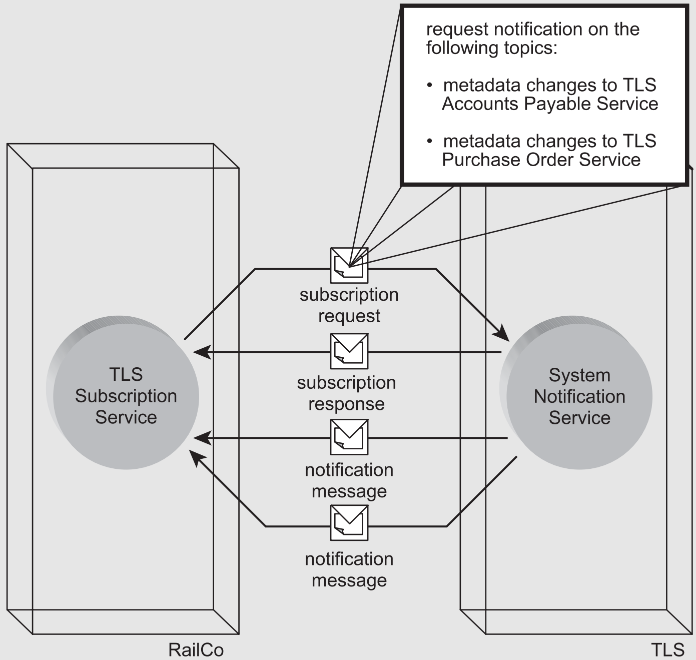

Figure 7.42

The new RailCo subscription service allows RailCo to receive notifications from the TLS System Notification Service.

SUMMARY OF KEY POINTS

- The traditional publish-and-subscribe messaging model can be implemented with the WS-Notification framework or the WS-Eventing specification.
- WS-Notification consists of the WS-BaseNotification, WS-Topics, and WS-Brokered-Notification specifications that collectively establish a subscription and notification system.
- The WS-Eventing specification provides similar functionality but is based on a moderately different architecture.
- Notification and eventing realize the popular publish-and-subscribe messaging model within SOA. The sophisticated messaging environment provided by SOA, in turn, introduces new opportunities to leverage these notification mechanisms.

Index

A

L

M

T

U

V

W