Weer_CH10.qxd 3/1/05 12:25 PM Page 187

Chapter 10

Reliable Messaging

Since the early 1990s, the information technology (IT) community has leveraged reliable messaging as a means of mitigating the issues presented in the scenarios and the motivations covered in this chapter. The IT community has been using message queue technologies such as WebSphereMQ from IBM, SonicMQ from Sonic, and MSMQ from Microsoft, in addition to reliable publish/subscribe technologies such as Tibco Rendezvous. The Java Community Process (JCP) has developed the Java Message Service API (JMS) in an effort to unify the myriad application programming interfaces (APIs) to these proprietary environments for the Java platform. These entities have adapted many of these reliable messaging environments for use in a Web services context by enabling them to carry SOAP messages and by describing their bindings using Web Services Description Language (WSDL). However, to date, each vendor tends to exploit its own proprietary protocol for the transmission of the message between its source and its destination. It's only possible to achieve interoperability between these proprietary messaging environments by means of gateways between disparate environments, each tailored to a specific pair of environments.

With the emergence of Web services as the preferred integration solution for distributed systems, it is now realistic to think about the possibility of a unified interoperability standard for reliable messaging.

Weer_CH10.qxd 3/1/05 12:25 PM Page 188

WS-Reliable Messaging has the greatest potential for becoming the standard for reliable messaging for Web services. Therefore, this book focuses on that specification.

10.1 Motivation for Reliable Messaging

L. Peter Deutsch, a noted computer scientist, has been attributed with publishing what has become known in software engineering circles as the "Eight Fallacies of Distributed Computing." He first presented them at a talk he gave to the researchers and engineers at Sun Microsystems Labs in 1991. At the time Deutsch first presented the fallacies, there were only seven. He added the eighth sometime later. The eight fallacies are as follows:

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogenous.

Web services are, at their essence, distributed applications. Certainly, when designing any Web service, you should carefully consider these words of wisdom. Ask yourself whether you have inadvertently relied upon any of these false assumptions in making your design decisions.

The next sections dive deeper into a few of these fallacies and discuss their relevance to Web services.

Weer_CH10.qxd 3/1/05 12:25 PM Page 189

10.1.1 The Network Is Reliable

The first of these fallacies, "The network is reliable," is one trap into which many software engineering projects fall. Given that most Web services deployed today use Transmission Control Protocol (TCP), a highly reliable connection-oriented host-to-host network protocol, you might think that it's unimportant to concern yourself with the inherent unreliability of the network. However, TCP is only reliable to the extent that the sending TCP stack can be certain that a message has been delivered to the TCP stack at the receiving host. Likewise, the receiving host can only be certain that it has either received a message reliably, or it has not. Things can still go wrong from the perspective of the Web service, which resides far above the TCP/IP interface.

First, consider that the reliability of the TCP protocol is limited in its scope to the two communicating TCP stacks and everything in between. Although the receiving TCP stack assumes responsibility for ensuring that received messages are passed to the application layer, the process could terminate before the TCP stack has been able to perform its responsibilities in this regard. Messages that have been successfully received and acknowledged at the TCP layer could be lost from the perspective of the application.

Second, a sending process might terminate before the sending application knows that the receiving TCP stack has received and acknowledged its message, and the receiving application has processed it.

Either of these two failure modes can leave a Web service consumer or provider in an inconsistent state with respect to its counterpart. Although this might not present a problem for certain stateless and/or idempotent operations such as an HTTP GET, it can present quite a serious problem for others that are not idempotent.

If you are going to provide for reliable messaging in the context of Web services, you need to keep these issues in mind.

Weer_CH10.qxd 3/1/05 12:25 PM Page 190

10.1.2 Latency Is Zero

Whether dealing with distributed components of an application on an intranet or over the Internet, latency between the distributed components impacts reliability. In the time it takes a message to be transmitted from sender to receiver, all manner of things can go wrong. The network could become partitioned due to a router failure or a severed or disconnected network cable. The destination host could crash. The process in which the receiving component is running could terminate.

When considering latency in terms of a round-trip request and response (or stimulus and response), latency becomes an even greater concern. If the service provider is overwhelmed with requests to process, you can often count latency in seconds, if not minutes. Processing a request can involve significant computational resources, or it might depend on another distributed component. Processing a request might even require manual intervention in some cases. The longer the latency, the greater the potential for something to go wrong, leaving the distributed application in an inconsistent state.

10.1.3 There Is One Administrator

Even in an intranet context, this fallacy often rears its ugly head. Although your IT department might assign a single group to be responsible for the network, in the context of a Web service, many administrators typically exist. In most cases, these administrators have rather parochial interests. There might be one administrator for each database, one for each of the application servers that host the Web service components, one for the demilitarized zone (DMZ) and firewall complexes, one for the server room, and so on. Administrators might not always coordinate their activities with your Web service's needs. All of this can lead to circumstances in which certain components of a Web service implementation become unavailable (during an upgrade or routine maintenance, for example), often at critical and unexpected times.

Expanding the scope of Web services to the context of the Internet, things get even more interesting and complicated. You can no more expect to coordinate activities related to the components of a Web service when the administrator(s) of those components are employed by your business partners than you can expect to win the lottery!

Weer_CH10.qxd 3/1/05 12:25 PM Page 191

Therefore, you need to design your Web service so that it can recover from failures related to the unavailability of a distributed component brought down for routine maintenance or failure.

10.2 Reliable Messaging Scenarios

Considering the issues highlighted by the previous elaborated fallacies, various strategies are available for designing a more robust and reliable system.

10.2.1 Store and Forward

In many scenarios in highly distributed systems, the best you can hope to achieve is to get information closer to its intended destination. This allows for more efficient use of the network resources and decouples the initial sender and intended recipient such that they do not need to be running concurrently. In other words, the sending system does not need to be running when the receiving node receives and processes the information, and the receiving node does not need to be running when the sending node transmits the information in the first place.

E-mail is at its essence a store-and-forward system. When you send an e-mail, it typically is transmitted first to your Internet service provider (ISP), where it is stored on disk. Then the ISP's sendmail server transmits the message to a server that is closer yet to the intended recipient (for example, the sendmail server at the intended recipient's ISP). Eventually, the intended recipient logs into her ISP and retrieves all new e-mail messages that have arrived. If any of the distributed components along the e-mail's message path is unavailable, the node at which the message is currently located attempts to retransmit the message later until such time as the message transmission, or hop, is successful.

10.2.2 Batch Window

This scenario is a derivative of the store-and-forward scenario discussed in the previous section. Consider two trading partners who want to exchange purchase orders reliably. One uses a batch system to process orders, and the other uses an online system that processes them in near real time. Clearly, there's an impedance mismatch between the two partners' systems. The partner

Weer_CH10.qxd 3/1/05 12:25 PM Page 192

with the batch order processing system might be in no position to rip and replace its existing batch system for any number of valid reasons, both business and technical.

An effective strategy for dealing with this impedance mismatch is to interpose a store-and-forward architecture between the two systems. You can employ this architecture at either partner or at both. The impedance mismatch is mitigated by virtue of the fact that the messages transmitted as individual messages can be collected into a batch for processing. The converse is also true. Neither side needs to be concerned with the fact that its counterpart has a different processing design.

10.2.3 Failure Recovery

Another scenario is that of failure recovery. Consider two trading partners that have been exchanging order information over time. One suffers a catastrophic failure of its order management system and has to recover from the previous evening's backup. That partner has lost all the orders that it has received since the backup from which it recovered was performed, and it must find some way of resynchronizing its system with that of its trading partner. Similarly, the partner whose system did not fail has an interest in having its trading partner fulfill all the orders that it has sent. Although the two partners could manually reconcile their systems, a manual reconciliation does not scale to scenarios that involve several trading partners. Had the partners been using a protocol that reliably exchanged messages that leveraged a persistent store and a capability to redeliver messages that had previously been acknowledged, they might be able to recover from such a failure in an automated manner, saving time and money in the process.

10.2.4 Long-Running Transactions

People often use atomic transactions with two or or three-phase commit strategies when designing distributed systems to ensure that the distributed components of the system have a consistent view of their shared state.

However, in many scenarios, use of a transaction processing monitor is not practical. You would never consider locking resources for extended periods necessary for completion of something as long running as a purchase order life cycle.

Weer_CH10.qxd 3/1/05 12:25 PM Page 193

Further, because of security considerations, you would never think twice about letting an external party place locks on your system's resources; that party could easily mount a denial-of-service (DoS) attack that could cripple your systems.

Using a reliable messaging protocol, paired with a protocol such as WS-Business Activity, to manage the compensation of application-level faults, you can overcome these concerns while enabling a more reliable interaction. The reliable messaging protocol ensures the dependable delivery of the messages, and the application-level compensation ensures that the long-running transaction either completes successfully or suitably compensates any failure.

10.3 Architectural Concepts

Of all the proposals for reliable messaging for Web services, WS-Reliable Messaging seems to have the most promise for broad adoption and widespread deployment necessary to become a standard for interoperability. That is because its sponsors represent a majority market-share in both the reliable messaging and Web services solution space.

WS-Reliable Messaging is by far the simplest of the proposed specifications. It focuses exclusively on the reliable messaging aspect. The addressing aspect has been factored out into a separate specification called WS-Addressing, covered in detail in Chapter 5, "WS-Addressing."

Reliable messaging is enabled by virtue of something called a *Sequence*, which is effectively a shared context for a set of messages to be delivered with a common quality of service between a sending and a receiving endpoint. Each message within a *Sequence* is assigned a unique message number, starting with 1 and increasing monotonically, by one, for each subsequent message in the Sequence. The receiving endpoint acknowledges receipt of the messages within a Sequence by indicating the range of messages it has received using a *SequenceAcknowledgement*. Each SequenceAcknowledgement message carries the acknowledgement information for all the messages that have been received within a Sequence. Hence, a SequenceAcknowledgement message does not require retransmission should it fail to reach the sending endpoint of the original message, because the information is sent with a subsequent SequenceAcknowledgement message.

Weer_CH10.qxd 3/1/05 12:25 PM Page 194

As with the other reliable messaging specifications proposed, WS-Reliable Messaging is defined as a set of SOAP Header extension elements that enable a range of qualities of service for a Web service, from at-most-once through exactly-once delivery assurances, preservation of message order, and duplicate detection. However, unlike the other proposals, WS-Reliable Messaging accomplishes this with a much simpler syntax and more efficient processing semantics.

10.4 Processing Model

The WS-Reliable Messaging specification defines an abstract model for the protocol and four distinct roles: Application Source and RM Source on the sending endpoint, and Application Destination and RM Destination at the receiving endpoint.

The Application Source role is typically played by the application code running on the endpoint from which messages are to be delivered reliably. It initiates the protocol by sending a message (logically) to the RM Source, which then assumes responsibility for transmitting—and possibly retransmitting—the message to the destination role at the receiving endpoint. The RM Source is also responsible for processing any SequenceAcknowledgement messages from the RM Destination and taking appropriate action.

The RM Destination role at the receiving endpoint receives messages (re)transmitted by the RM Source role at the sending endpoint. It is responsible for acknowledging receipt of the message and (logically) delivering the message to the Application Destination role, which is typically played by the application code that runs on the receiving endpoint. The RM Destination role is responsible for affecting the quality of service associated with the Delivery Assurance policy specified for the receiving endpoint.

Figure 10-1 demonstrates this model.

Weer_CH10.qxd 3/1/05 12:25 PM Page 195

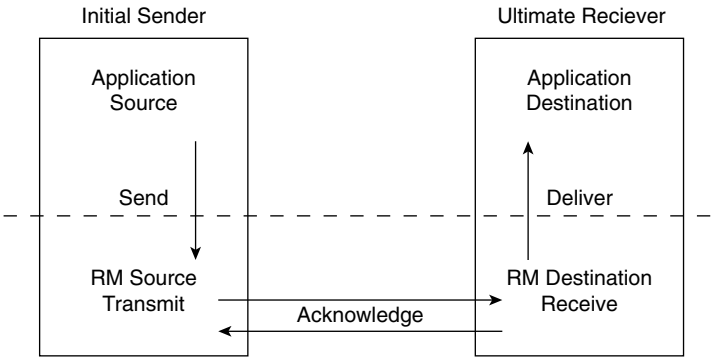

Figure 10-1 Reliable Messaging model.

This model is like that of the TCP protocol. The WS-Reliable Messaging protocol is limited in its scope to the RM Source and RM Destination roles at either endpoint. Aside from the assurances that the RM Destination observes to fulfill the specified delivery assurances (At-Most-Once, At-Least-Once, Exactly-Once, and Ordered), the Application Source role can only be certain that the message has been reliably delivered to the RM Destination role at the receiving endpoint.

If the application code at the Application Source role needs to have some sort of application-level acknowledgement that the message was actually processed, the WS-Reliable Messaging protocol is inadequate for the task. An application-specific acknowledgement message is required, such as a PurchaseOrderAck as an application-level acknowledgement/response to a submitted PurchaseOrder.

Note, however, that this separation of concerns is important. Examine the previous example in light of this separation of concerns. Consider that a PurchaseOrderAck message sent in response to a PurchaseOrder submission message might take upward of 24 hours to process in the use case, where the receiving partner processes its received PurchaseOrders in a batch window once a day. Waiting 24 hours for some indication that a PurchaseOrder has been lost in transit between the sending and receiving endpoints can have serious consequences for the business partner that makes the request. It can even lead to loss of revenue when order fulfillment is time-sensitive. Clearly, relying exclusively on the application-level acknowledgement as an indicator that the message was successfully transmitted is not an ideal situation for all use cases.

Weer_CH10.qxd 3/1/05 12:25 PM Page 196

Leveraging an infrastructure-level (or, more precisely, a middleware-level) acknowledgement, decoupled from the application-level processing of the message, as the indicator of successful transmission of a message enables the sending endpoint to respond more quickly to failed transmission attempts. The failed message transmission can be retried until its receipt has been acknowledged. The receiving software can take the necessary steps to ensure that the message is eventually delivered to the application by means of a persistent store or message queue that the receiving application accesses when it is ready and able.

This model is highly effective at mitigating the issues discussed earlier in the "Motivation for Reliable Messaging" section.

The ensuing sections discuss how the WS-Reliable Messaging protocol realizes this model.

10.4.1 Sequence Lifecycle

An RM Sequence has a well-defined lifecycle. It begins with the RM Source requesting the creation of a new Sequence of the RM Destination using the CreateSequence operation. The following example demonstrates the CreateSequence message.

```
<?xml version="1.0" encoding="UTF-8"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
 <soap:Header>
  <wsa:MessageID>
   http://example.com/guid/0baaf88d-483b-4ecf-a6d8-a7c2eb546817
  </wsa:MessageID>
  <wsa:To>http://example.com/service/B</wsa:To>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2005/02/rm/CreateSequence
    </wsa:Action>
  <wsa:ReplyTo>
   <wsa:Address>http://example.com/service/A</wsa:Address>
  </wsa:ReplyTo>
 </soap:Header>
```

Weer_CH10.qxd 3/1/05 12:25 PM Page 197


```
 <soap:Body>
  <wsrm:CreateSequence>
    <wsrm:AcksTo>
      <wsa:Address>http://example.com/service/A</wsa:Address>
    </wsrm:AcksTo>
  </wsrm:CreateSequence>
 </soap:Body>
</soap:Envelope>
```

The CreateSequence element has a single required child element, the AcksTo, which indicates the WS-Addressing endpoint to which SequenceAcknowledgement messages are to be delivered.

Upon receipt of the CreateSequence message, the RM Destination responds by creating a new Sequence, assigning it a unique identifier, and by returning a CreateSequenceResponse message, containing the Sequence's identifier, to the RM Source. The following example demonstrates a CreateSequenceResponse message.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <soap:Header>
    <wsa:MessageID>
   http://example.com.com/guid/0baaf88d-483b-4ecf-a6d8-
a7c2eb546818
  </wsa:MessageID>
    <wsa:To>http://example.com/service/A</wsa:To>
    <wsa:RelatesTo>
      http://example.com/guid/0baaf88d-483b-4ecf-a6d8a7c2eb546817
    </wsa:RelatesTo>
    <wsa:Action>

http://schemas.xmlsoap.org/ws/2005/02/rm/CreateSequenceResponse
    </wsa:Action>
  </soap:Header>
  <soap:Body>
    <wsrm:CreateSequenceResponse>

<wsrm:Identifier>http://example.com/RM/ABC</wsrm:Identifier>
    </wsrm:CreateSequenceResponse>
  </soap:Body>
</soap:Envelope>
```

Weer_CH10.qxd 3/1/05 12:25 PM Page 198

The Sequence then transmits messages between the RM Source and the RM Destination. When all of the messages in a Sequence have been transmitted and successfully acknowledged, the Sequence is terminated. Upon receipt of the SequenceAcknowledgement that acknowledges the entire range of messages in the Sequence, the RM Source sends a TerminateSequence one-way message to the RM Destination. Upon receipt of the TerminateSequence message, the RM Destination is free to reclaim any resources that are associated with the Sequence. It can do so because it can be assured that the RM Source has received the final SequenceAcknowledgement message covering the full range of messages in the Sequence and will be sending no further messages in the Sequence.

If the RM Destination receives any subsequent messages in the Sequence, it can be assured that these messages must have been caught in the network and can be safely discarded without action. In fact, if the RM Destination receives any message belonging to a Sequence about which it has no knowledge, it can safely discard the message without taking action because it can assume that the message belongs to a Sequence that has been terminated.

If the RM Destination does not receive the TerminateSequence message, it may preserve the state associated with the Sequence until the Sequence expiry duration expires so that it can respond with a retransmission of the final SequenceAcknowledgement message if any subsequent messages for the Sequence are received.

10.4.2 Basic Syntax

The WS-Reliable Messaging specification defines four SOAP header elements: Sequence, SequenceFault, SequenceAcknowledgement, and AckRequested. In addition, a companion specification, WS-RM Policy Assertion, defines a set of domain-specific policy assertions, to be used in context of WS-Policy and WS-Policy Attachments for purposes of specifying the quality of service details related to a Sequence.

The sections that follow explore each of the syntax elements in detail.

Weer_CH10.qxd 3/1/05 12:25 PM Page 199

10.4.3 Sequence Element

The core element of the WS-Reliable Messaging protocol is the Sequence element. Each message within a Sequence between sending and receiving endpoints must include a Sequence SOAP header element. The following is an example of a SOAP message containing a Sequence header that initiates a new Sequence and establishes the initial expiration date for the Sequence:

```
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
 xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm">
  <soap:Header>
    <rm:Sequence>
      <rm:Identifier>
        20030818-11010001-0500@example.com
      </rm:Identifier>
      <rm:MessageNumber>1</rm:MessageNumber>
    </rm:Sequence>
  </soap:Header>
  <soap:Body>
    ...
  </soap:Body>
</soap:Envelope>
```

The Sequence element has three child elements: Identifier, MessageNumber, and LastMessage. The Identifier and MessageNumber elements must be present in each Sequence element.

The Identifier element is the unique identifier of the Sequence. The same value must be present on each message in a Sequence.

The MessageNumber element carries a positive integer that represents the message's position within a Sequence.

The LastMessage element is included only on the last message in a Sequence. It signals to the receiving endpoint that this message is the last in the Sequence.

Weer_CH10.qxd 3/1/05 12:25 PM Page 200

In addition, you can extend the Sequence element so that it can carry additional attributes and elements from foreign namespaces. This allows the Sequence element to be composed in several ways, some of which the authors of the WS-Reliable Messaging specification might not have anticipated.

A Sequence can be short-lived or long-running. It can have as few as one message or as many as 18,446,744,073,709,551,615 messages. Therefore, you can leverage the protocol for a wide variety of use cases. Partners who exchange relatively few messages at infrequent intervals might choose to establish a new Sequence for each burst of activity, whereas partners who have long-term relationships with frequent exchanges of messages might choose to do so under the context of a single Sequence. In either case, the protocol is always the same.

A Sequence can apply to all of the messages traveling in a particular direction (for example, from sender to receiver) in a portType/interface, to messages traveling in a particular direction within selected operations in a portType/interface, or to particular input or output messages traveling in a particular direction. Technically, a Sequence can apply to all traffic traveling in a particular direction between two endpoints, regardless of whether they have been described as a single portType/interface.

10.4.4 SequenceAcknowledgement Element

Messages within a Sequence are acknowledged by a receiving endpoint by means of the SequenceAcknowledgement SOAP header element. As previously mentioned, the SequenceAcknowledgement SOAP header element contains acknowledgement information about all of the messages in a Sequence. It does this by means of a set of one or more child AcknowledgementRange elements, each of which carries an upper and lower bound of contiguous messages that have been received within a Sequence.

Technically, the correctness of the protocol can be accomplished by virtue of the receiving endpoint sending just one acknowledgement message with a SequenceAcknowledgement element in response to the receipt of a message that has a Sequence element with the LastMessage element present.

Weer_CH10.qxd 3/1/05 12:25 PM Page 201

Of course, waiting until all the messages within a Sequence have been received before sending an acknowledgement message might be impractical in many, if not most, circumstances. As you will soon see in the section titled "10.4.8 Policy Assertions," a receiving endpoint is typically configured with a maximum interval between acknowledgement messages using the AcknowledgementInterval policy assertion. A sending endpoint then expects to receive an acknowledgement message within the specified AcknowledgmentInterval after transmitting a message. Failure to receive a SequenceAcknowledgement message within the specified time interval might indicate a problem at the receiving endpoint and result in a retransmission attempt for any unacknowledged messages.

The following is an example of a SequenceAcknowledgement element that acknowledges receipt of messages 1 through 10.

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm">
  <soap:Header>
    ...
    <rm:SequenceAcknowledgment>
      <rm:Identifier>
       20030818-11010001-0500@example.com
      </rm:Identifier>
      <rm:AcknowledgmentRange Upper="10" Lower="1"/>
    </rm:SequenceAcknowledgment>
  </soap:Header>
  <soap:Body>
    ...
  </soap:Body>
</soap:Envelope>
```

Next is an example of a SOAP message that contains a Sequence Acknowledgement SOAP header element. This element indicates that it has not received the third message in a Sequence that has five messages (as perceived from the receiving endpoint's perspective). Note that the SequenceAcknowledgement element has two AcknowledgementRange child elements: one that acknowledges receipt of messages 1 and 2, and a second that acknowledges receipt of messages 4 and 5.

Weer_CH10.qxd 3/1/05 12:25 PM Page 202


```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm">
  <soap:Header>
    <rm:SequenceAcknowledgment>
      <rm:Identifier>
       20030818-11010001-0500@example.com
    </rm:Identifier>
      <rm:AcknowledgmentRange Upper="2" Lower="1"/>
      <rm:AcknowledgmentRange Upper="5" Lower="4"/>
    </rm:SequenceAcknowledgment>
  </soap:Header>
  <soap:Body>
    ...
  </soap:Body>
</soap:Envelope>
```

The sending endpoint that received such a SequenceAcknowledgement would determine that message 3 needed to be retransmitted until the sending endpoint received a message carrying a SequenceAcknowledgement SOAP header element indicating that the receiving endpoint had, in fact, received message 3.

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm">
  <soap:Header>
    ...
    <rm:SequenceAcknowledgment>
      <rm:Identifier>20030818-11010001-
       0500@example.com</rm:Identifier>
      <rm:AcknowledgmentRange Upper="5" Lower="1"/>
    </rm:SequenceAcknowledgment>
  </soap:Header>
  <soap:Body>
    ...
  </soap:Body>
</soap:Envelope>
```

10.4.5 AckRequested Element

The third SOAP header element defined by the WS-Reliable Messaging specification is the AckRequested element. A sending endpoint might include an AckRequested SOAP header element in a message as a means of asking the receiving endpoint to send an acknowledgement message immediately instead of waiting for the acknowledgement interval to expire.

Weer_CH10.qxd 3/1/05 12:25 PM Page 203

The following SOAP message is an example of use of the AckRequested element.

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
  xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm"
  xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
  <soap:Header>
    ...
    <rm:Sequence>
      <rm:Identifier>20030818-11010001-
       0500@example.com</rm:Identifier>
      <rm:MessageNumber>1</rm:MessageNumber>
    </rm:Sequence>
    <rm:AckRequested/>
  </soap:Header>
  <soap:Body>
    ...
  </soap:Body>
</soap:Envelope>
```

10.4.6 SequenceFault Element

The fourth and final SOAP header element defined in the WS-Reliable Messaging specification is SequenceFault. It carries fault detail information related to the processing of the Sequence, SequenceAcknowledgement, and AckRequested SOAP header elements. The fault information is carried as a SOAP header element because faults that are related to the processing of SOAP header elements must also be conveyed as SOAP header elements per the SOAP specification(s).

The SequenceFault element has two required child elements: Identifier, to identify the Sequence to which the fault applies, and FaultCode, to carry the qualified name (QName) of one of the fault codes defined in the WS-Reliable Messaging specification. It might also have one or more AcknowledgementRange child elements if the fault was generated in response to processing a SequenceAcknowledgement.

In addition, the specification provides for extensibility of the SequenceFault element by means of an XML Schema wildcard element.

Weer_CH10.qxd 3/1/05 12:25 PM Page 204

The following is an example of a SOAP message carrying a SequenceFault SOAP header element. It indicates that the Sequence has been refused because of insufficient resources at the RM Destination to allocate a new Sequence.

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm"
  xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
  <soap:Header>
    <rm:SequenceFault>
      <rm:FaultCode>rm:SequenceRefused</rm:FaultCode>
    </rm:SequenceFault>
  </soap:Header>
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Server</faultcode>
      <faultstring>
      insufficient resources for new Sequence.
      </faultstring>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

10.4.7 Delivery Semantics Supported

The WS-Reliable Messaging protocol is an At-Least-Once protocol. That means that each message will be delivered to the RM Destination at least once in the correct operation of the protocol. This base enables the RM Destination to offer the full spectrum of delivery assurances to the Application Destination, ranging from At-Most-Once to At-Least-Once and Exactly-Once to Ordered.

For instance, the RM Destination could maintain a limited buffer for messages received and discard duplicates and the oldest undelivered messages in the buffer as new messages are received within a Sequence after the buffer size is exceeded. That would provide At-Most-Once delivery semantics to the Application Destination.

Alternatively, the RM Destination could be implemented such that it did not check for and discard duplicate messages, thus providing At-Least-Once delivery semantics to the Application Destination.

Weer_CH10.qxd 3/1/05 12:25 PM Page 205

The RM Destination could apply Exactly-Once delivery semantics by applying duplicate detection to the At-Least-Once semantics.

The RM Destination could provide "Ordered" delivery semantics by ensuring that it buffered those messages that have gaps in the Sequence until messages are received that fill those gaps before delivering the messages to the Application Destination.

Ultimately, it is the RM Destination's responsibility to fulfill the delivery assurance requirements of the Application Destination. This simplifies the protocol and increases the potential for interoperability.

10.4.8 Policy Assertions

The companion WS-RM Policy Assertion specification defines a policy assertion that is intended for use in the context of WS-Policy and WS-Policy Attachments. The policy assertion is aimed at enabling endpoints that participate in the WS-Reliable Messaging protocol either to specify their requirements or to indicate the protocol's observed behavior to a prospective partner.

10.4.9 Inactivity Timeout

This assertion property specifies (in milliseconds) a period of inactivity for a Sequence. If during this duration an endpoint has received no application or control messages, the endpoint MAY consider the Sequence to have been terminated due to inactivity.

```
<rm:InactivityTimeout Milliseconds="86400000"/>
```

10.4.10 Retransmission Interval

A ReliableMessaging source may optionally specify a base retransmission interval for a sequence. If no acknowledgement has been received for a given message within that interval, the source will retransmit the message. The retransmission interval may be modified at the discretion of the source during the lifetime of the sequence. This assertion property does not alter the formulation of messages as transmitted, only the timing of their transmission.

Weer_CH10.qxd 3/1/05 12:25 PM Page 206

The sequence may optionally specify that the interval will be adjusted using the commonly known exponential backoff algorithm.

```
<rm:BaseRetransmissionInterval Milliseconds="3000"/>
<rm:ExponentialBackoff/>
```

10.4.11 Acknowledgement Interval

Acknowledgements can be sent on return messages or sent stand alone. In the case where a return message is not available with which to send an acknowledgement, a ReliableMessaging Destination may wait for the duration of the acknowledgement interval before sending a stand alone acknowledgement. If there are no unacknowledged messages, the ReliableMessaging Destination may choose not to send an acknowledgement.

This assertion property does not alter the formulation of messages or acknowledgements as transmitted. Its purpose is to communicate the timing of acknowledgements so that the source may be tuned appropriately. It does not alter the meaning of the <AckRequested> directive.

```
<rm:AcknowledgementInterval Milliseconds="1000"/>
```

10.4.12 Basic WS-Reliable Messaging Profile

Given that the policy assertion properties that comprise timing considerations can be arranged into any number of possible combinations, the authors of the WS-Reliable Messaging specification have defined a base timing profile that is intended to help promote interoperability.

The specifics of this profile are as follows:

```
<wsp:Policy>
  <rm:RMAssertion>
    <rm:BaseRetransmissionInterval Milliseconds="3000"/>
    <rm:ExponentialBackoff/>
    <rm:InactivityTimeout Milliseconds="86400000"/>
    <rm:AcknowledgementInterval Milliseconds="1000"/>
  </rm:RMAssertion>
</wsp:Policy>
```

Weer_CH10.qxd 3/1/05 12:25 PM Page 207

The authors have assigned this profile the following URI designation:

```
http://schemas.xmlsoap.org/ws/2005/02/rm/baseTimingProfile.xml
```

Therefore, you can reference it using a WS-Policy Attachment PolicyReference.

10.5 Strengths and Weaknesses

On the plus side, WS-Reliable Messaging offers the full range of quality of service that you would expect. The protocol is quite simple, but in many respects, it is far more efficient and effective than the other proposed specifications.

WS-Reliable Messaging has been carefully architected to be fully composeable with other Web services specifications that have been or have yet to be published by IBM, Microsoft, and their partners.

Specifically, WS-Reliable Messaging can be composed with WS-Addressing to enable a wide variety of reliable message exchange patterns (MEPs). However, WS-Reliable Messaging does not require composition with WS-Addressing or any specific version of WS-Addressing. For example, consider the case in which WS-Reliable Messaging is used with the synchronous SOAP/HTTP binding, such that the SequenceAcknowledgment is carried in a SOAP message on the HTTP response message. Typically, there would be no need to address information in this use case. Yet, it enables a far more robust and reliable exchange of messages between the sending and receiving endpoints. If a SequenceAcknowledgement message fails to reach the sending endpoint that initiated the HTTP request, the acknowledgement information is carried on the HTTP response for the next message in the Sequence without requiring that the sending endpoint resend the unacknowledged message.

WS-Reliable Messaging does not require specialized logic to validate required interdependencies between header elements that cannot be expressed in XML Schema, as do some of the other two proposed specifications that this chapter has reviewed. Therefore, off-the-shelf schema validators, such as the Apache Xerces parser, can validate the SOAP header elements.

Weer_CH10.qxd 3/1/05 12:25 PM Page 208

WS-Reliable Messaging has also been carefully designed to be extensible. It can add optional extension element and attribute content from a foreign namespace in a manner that does not require implementations to be upgraded but allows those that do upgrade to take advantage of the extended features.

10.6 Examples

The subsequent examples demonstrate a typical message exchange using the WS-Reliable Messaging protocol. What follows is the initial message to create the Sequence using a CreateSequence message.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
<soap:Header>
  <wsa:MessageID>
   http://example.com.com/0baaf88d-483b-4ecf-a6d8-a7c2eb546817
  </wsa:MessageID>
  <wsa:To>http://example.com.com/service/B</wsa:To>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2005/02/rm/CreateSequence
    </wsa:Action>
  <wsa:ReplyTo>
<wsa:Address>http://example.com.com/service/A</wsa:Address>
  </wsa:ReplyTo>
 </soap:Header>
 <soap:Body>
  <wsrm:CreateSequence>
    <wsrm:AcksTo>
      <wsa:Address>
        http://example.com.com/service/A
      </wsa:Address>
    </wsrm:AcksTo>
  </wsrm:CreateSequence>
 </soap:Body>
</soap:Envelope>
```

Next is a CreateSequenceResponse message.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm"
```

Weer_CH10.qxd 3/1/05 12:25 PM Page 209


```
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <soap:Header>
    <wsa:MessageID>
   http://example.com.com/guid/0baaf88d-483b-4ecf-a6d8-a7c2eb546818
  </wsa:MessageID>
    <wsa:To>http://example.com.com/service/A</wsa:To>
    <wsa:RelatesTo>
      http://example.com.com/guid/0baaf88d-483b-4ecf-a6d8a7c2
➥eb546817
    </wsa:RelatesTo>
    <wsa:Action>

http://schemas.xmlsoap.org/ws/2005/02/rm/CreateSequenceResponse
    </wsa:Action>
  </soap:Header>
  <soap:Body>
    <wsrm:CreateSequenceResponse>
      <wsrm:Identifier>
        http://example.com.com/RM/ABC
      </wsrm:Identifier>
    </wsrm:CreateSequenceResponse>
  </soap:Body>
</soap:Envelope>
```

After WS-Reliable Messaging creates the Sequence, messages can begin flowing for the Sequence. In the next example, the Sequence has three messages, and one of the messages is lost in transit, requiring a retransmission.

Message 1

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <soap:Header>
    <wsa:MessageID>
      http://example.com.com/guid/71e0654e-5ce8-477b-bb9d-34f05
➥cfcbc9e
    </wsa:MessageID>
    <wsa:To>http://example.com.com/service/B</wsa:To>
    <wsa:From>
      <wsa:Address>
        http://example.com.com/service/A
      </wsa:Address>
    </wsa:From>
    <wsa:Action>
      http://example.com.com/service/B/request
    </wsa:Action>
```

Weer_CH10.qxd 3/1/05 12:25 PM Page 210


```
    <wsrm:Sequence>
      <wsrm:Identifier>
        http://example.com.com/RM/ABC
      </wsrm:Identifier>
      <wsrm:MessageNumber>1</wsrm:MessageNumber>
    </wsrm:Sequence>
  </soap:Header>
  <soap:Body>
    <!—  Some  Application  Data  —>
  </soap:Body>
</soap:Envelope>
```

Message 2

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <soap:Header>
    <wsa:MessageID>
      http://example.com.com/guid/daa7d0b2-c8e0-476e-a9a4-d164154
➥e38de
    </wsa:MessageID>
    <wsa:To>http://example.com.com/service/B</wsa:To>
    <wsa:From>
      <wsa:Address>
        http://example.com.com/service/A
      </wsa:Address>
    </wsa:From>
    <wsa:Action>
      http://example.com.com/service/B/request
    </wsa:Action>
    <wsrm:Sequence>
      <wsrm:Identifier>
        http://example.com.com/RM/ABC
      </wsrm:Identifier>
      <wsrm:MessageNumber>2</wsrm:MessageNumber>
    </wsrm:Sequence>
  </soap:Header>
  <soap:Body>
    <!—  Some  Application  Data  —>
  </soap:Body>
</soap:Envelope>
```

Message 3

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
```

Weer_CH10.qxd 3/1/05 12:25 PM Page 211


```
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
 <soap:Header>
  <wsa:MessageID>
   http://example.com.com/guid/0baaf88d-483b-4ecf-a6d8-
➥a7c2eb546817
  </wsa:MessageID>
  <wsa:To>http://example.com.com/service/B</wsa:To>
  <wsa:From>
   <wsa:Address>
     http://example.com.com/service/A
   </wsa:Address>
  </wsa:From>
  <wsa:Action>
    http://example.com.com/service/B/request
  </wsa:Action>
  <wsrm:Sequence>
   <wsrm:Identifier>
    http://example.com.com/RM/ABC
   </wsrm:Identifier>
   <wsrm:MessageNumber>3</wsrm:MessageNumber>
   <wsrm:LastMessage/>
  </wsrm:Sequence>
 </soap:Header>
 <soap:Body>
  <!— Some Application Data —>
 </soap:Body>
</soap:Envelope>
```

The RM Destination has not received message 2 because of a transmission error, so it responds with a SequenceAcknowledgement for messages 1 and 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
 <soap:Header>
  <wsa:MessageID>
   http://example.com.com/guid/0baaf88d-483b-4ecf-a6d8-
➥a7c2eb546817
  </wsa:MessageID>
  <wsa:To>http://example.com.com/service/A</wsa:To>
  <wsa:From>
   <wsa:Address>http://example.com.com/service/B</wsa:Address>
  </wsa:From>
   <wsa:Address>http://example.com.com/service/B</wsa:Address>
  </wsa:From>
```

Weer_CH10.qxd 3/1/05 12:25 PM Page 212


```
  <wsa:RelatesTo>
    http://example.com.com/guid/0baaf88d-483b-4ecf-a6d8-
➥a7c2eb546817
  </wsa:RelatesTo>
  <wsa:Action>

http://schemas.xmlsoap.org/ws/2005/02/rm/SequenceAcknowledgement
  </wsa:Action>
  <wsrm:SequenceAcknowledgement>
   <wsrm:Identifier>
    http://example.com.com/RM/ABC
   </wsrm:Identifier>
   <wsrm:AcknowledgementRange Upper="1" Lower="1"/>
   <wsrm:AcknowledgementRange Upper="3" Lower="3"/>
  </wsrm:SequenceAcknowledgement>
 </soap:Header>
 <soap:Body/>
</soap:Envelope>
```

The sending endpoint discovers that the RM Destination did not receive message 2, so it resends the message and requests an acknowledgement.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
 <soap:Header>
  <wsa:MessageID>
   http://example.com.com/guid/0baaf88d-483b-4ecf-a6d8-
➥a7c2eb546817
  </wsa:MessageID>
  <wsa:To>http://example.com.com/service/B</wsa:To>
  <wsa:From>
   <wsa:Address>http://example.com.com/service/A</wsa:Address>
  </wsa:From>
  <wsrm:Sequence>
   <wsrm:Identifier>
    http://example.com.com/RM/ABC
   </wsrm:Identifier>
   <wsrm:MessageNumber>2</wsrm:MessageNumber>
  </wsrm:Sequence>
  <wsrm:AckRequested>
   <wsrm:Identifier>
    http://example.com.com/RM/ABC
   </wsrm:Identifier>
  </wsrm:AckRequested>
```

Weer_CH10.qxd 3/1/05 12:25 PM Page 213


```
 </soap:Header>
 <soap:Body>
  <!-- Some Application Data -->
 </soap:Body>
</soap:Envelope>
```

The RM Destination responds with a SequenceAcknowledgement for the complete sequence, which can then be terminated.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
 <soap:Header>
  <wsa:MessageID>
   http://example.com.com/guid/0baaf88d-483b-4ecf-a6d8-
➥a7c2eb546817
  </wsa:MessageID>
  <wsa:To>http://example.com.com/service/A</wsa:To>
  <wsa:From>
   <wsa:Address>http://example.com.com/service/B</wsa:Address>
  </wsa:From>
  <wsa:RelatesTo>
    http://example.com.com/guid/0baaf88d-483b-4ecf-a6d8-
➥a7c2eb546817
  </wsa:RelatesTo>
  <wsa:Action>

http://schemas.xmlsoap.org/ws/2005/02/rm/SequenceAcknowledgement
  </wsa:Action>
  <wsrm:SequenceAcknowledgement>
   <wsrm:Identifier>
    http://example.com.com/RM/ABC
   </wsrm:Identifier>
   <wsrm:AcknowledgementRange Upper="3" Lower="1"/>
  </wsrm:SequenceAcknowledgement>
 </soap:Header>
 <soap:Body/>
</soap:Envelope>
```

The RM Source receives the final SequenceAcknowledgement message. Then it sends a TerminateSequence to the RM Destination to indicate that it can reclaim the resources that are associated with the Sequence.

Weer_CH10.qxd 3/1/05 12:25 PM Page 214


```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
 <soap:Header>
  <wsa:MessageID>
   http://example.com.com/guid/0baaf88d-483b-4ecf-a6d8-
➥a7c2eb546817
  </wsa:MessageID>
  <wsa:To>http://example.com.com/service/B</wsa:To>
  <wsa:Action>
    http://schemas.xmlsoap.org/ws/2005/02/rm/TerminateSequence
  </wsa:Action>
  <wsa:From>
   <wsa:Address>http://example.com.com/service/A</wsa:Address>
  </wsa:From>
 </soap:Header>
 <soap:Body>
  <wsrm:TerminateSequence>
   <wsrm:Identifier>
    http://example.com.com/RM/ABC
   </wsrm:Identifier>
  </wsrm:TerminateSequence>
 </soap:Body>
</soap:Envelope>
```

10.7 Future Directions

At the time of this writing, the authors and others who have developed implementations are performing interoperability and composeability testing on the WS-Reliable Messaging specification. After the interoperability and composeability testing are complete, the developers will republish the specification and submit it to a standards body. They haven't chosen a venue yet.

The specification is expected to be leveraged so that it provides for a standards-based interoperability protocol to bridge the various proprietary JMS provider environments.

Weer_CH10.qxd 3/1/05 12:25 PM Page 215

10.8 Summary

This chapter covered the motivations for reliable messaging, including some of L. Peter Deutsch's "Eight Fallacies of Distributed Computing." Reliable messaging has served IT well as a foundation for many enterprise application integration deployments, providing the loose coupling necessary to mitigate against those fallacies. WS-Reliable Messaging promises to bring the benefits of reliable messaging to SOA and Web services, enabling enterprises to extend Web services to support reliable business-to-business (B2B) exchanges. This would replace Electronic Data Interchange (EDI) and offer the potential for a single interoperability protocol for the various proprietary Message Oriented Middleware (MOM) protocols.

Weer_CH10.qxd 3/1/05 12:25 PM Page 216