Using Message-oriented Middleware for Reliable Web Services Messaging

Stefan Tai, Thomas A. Mikalsen, Isabelle Rouvellou

IBM T.J. Watson Research Center, Hawthorne, New York, USA
{stai, tommi, rouvellou}@us.ibm.com

Abstract. Web Services hold the promise of a standards-based platform for automating the integration of applications over diverse networks, operating systems and programming languages. Reliable messaging is critical in this context; many enterprise systems require a messaging infrastructure that guarantees message delivery even in the presence of software and network failures. Using existing message-oriented middleware (MOM) for reliable Web services messaging seems natural. However, a variety of implementation challenges, including the support for specific reliable Web services messaging protocols, must be addressed. In this paper, we discuss the options for and implications of employing MOM to implement reliable messaging for Web services. In doing so, we contribute to the understanding of reliability for Web services in general.

1 Introduction

Web services are applications that are described, published, and accessed over the Web using open XML standards. They promote a service-oriented computing model where an application exposes, using the Web Services Description Language (WSDL) [8] both its functionality (in a platform independent fashion) as well as its mappings to available communication protocols and deployed service implementations. This description can be discovered by client applications using service registries in order for the client to then use the service by means of XML messaging.

A key goal of the Web services framework is to provide a standards-based abstraction layer over diverse network transports, operating systems, and programming languages, and therefore, to provide a platform for automating the integration of these diverse systems and their applications. Further, the framework aims to support application/system integration both within and between organizational boundaries. As Web services reach their full potential, new services will emerge that are based on compositions of other Web services and specified using interoperable business process standards.

If this vision is to be realized, basic service interactions, spanning technological and organizational boundaries, must be reliable. Consequently, new protocols for reliable messaging for Web services have been proposed [3] [9] [22] [15]. Yet, the use of existing message-oriented middleware (MOM) for reliable Web services messaging seems reasonable, too. There are now many different options for

implementing reliable messaging for Web services, with various implications on overall application-to-application reliability.

This paper provides some clarity about reliable messaging and using message-oriented middleware for Web services. We first discuss different aspects of reliable messaging, identifying the facets of MOM that affect reliability. We then explore different, exemplary options for applying existing messaging technologies to achieve reliable messaging for Web services. We explain the options in detail, and assess them with respect to the level of reliability they achieve and the assumptions they place on the implementation of the distributed services.

2 Messaging Middleware Reliability

In distributed systems, multiple processes (executing on different nodes in a network) interact by sending and receiving messages. In this environment, it is often desirable to decouple the sending and receiving processes such that forward progress can be made in the presence of failures and temporary unavailability. To achieve this loose-coupling of processes, message-oriented middleware is often employed.

Messaging middleware is specialized software that accepts messages from sending processes and delivers them to receiving processes (typically across a network). Such middleware typically support two common delivery patterns: point-to-point (p2p) and publish/subscribe (pub/sub). The architecture of the middleware can take many forms. The two principle styles are centralized and distributed architectures. With centralized architectures, all processes communicate with a common messaging server. In distributed architectures, processes communicate with local messaging middleware components; these local messaging components then communicate over the network to deliver messages on behalf of senders and receivers. In this paper, we will focus on distributed architectures, as these are natural in a Web environment.

2.1 Aspects of Reliability

Messaging middleware can perform various functions that facilitate reliable, loose-coupling. In distributed architectures, the sender's messaging component can tolerate network failures by repeatedly sending a message until it is acknowledged by the receiver's messaging component; this interaction can occur even after the sending process has terminated (or is otherwise unavailable). The receiver's messaging component can tolerate the unavailability of the receiving process by maintaining messages until the receiving process is ready. From the sender application's viewpoint, this allows the application to "fire-and-forget" messages, relying on the middleware to guarantee message delivery.

The messaging endpoints ideally are the messaging clients (the sending and receiving applications). With common messaging middleware, the guarantees are, however, typically restricted to the middleware endpoints of message brokers and message queue managers. It is then assumed that a messaging client accesses a messaging middleware endpoint locally and using transactions (though a distributed

and/or non-transactional access is possible, too). The middleware only ensures message delivery within its own "network" of managerial messaging endpoints.

In addition to acknowledged delivery (through proper correlation of messages and acknowledgments), ordered delivery of messages is another aspect of reliable messaging middleware. This is particularly important in asynchronous environments, where messages are typically stored by the middleware (for example, a message queue) before they are dispatched to or retrieved by final recipients.

Messages can further be attributed with expiration timestamps, a priority attribute, a reply-to address, and other properties that contribute to messaging reliability. Such attributes are checked by the middleware in order to prevent delivery of a message if it is no longer valid, to prioritize messages that are stored at the middleware for later dispatch, and to guarantee that a receipt acknowledgment is sent back to a specific, application-defined reply address.

A further important aspect of reliability concerns the integration of a message delivery in a larger processing context. A message typically is part of some business process (messaging conversation) and atomic unit-of-work. Key requirements on reliable messaging middleware therefore include the ability to atomically group a message with other messages and other process activities, and, to integrate a message store like a message queue as a resource manager in a distribution transaction.

2.2 Three Facets of Reliability

The above described aspects of reliability lead to the definition of three general facets of reliability:

Middleware endpoint-to-endpoint reliability
A message, once delivered from an application (process) to the messaging middleware, is guaranteed to be (eventually) available for consumption by the receiving process. The middleware ensures eventual message delivery within its distributed network of middleware endpoints.

Application-to-middleware reliability
The middleware's messaging API, used to send and receive messages, supports reliability properties such as message delivery guarantees, message persistence, and transactional messaging.

Application-to-application reliability
Sending and receiving applications engage in transactional business processes that rely on application-to-middleware reliability and middleware endpoint-to-endpoint reliability.

2.3 Technologies

Each facet of reliable messaging imposes certain requirements on the middleware. Fundamental requirements include the integration of a persistent message store and the implementation of a reliable protocol to move messages between these persistent stores.

Examples of message-oriented middleware products implementing reliable messaging are IBM's Websphere MQ (formerly called MQSeries) [10], TIBCO's Rendezvous [21], and Microsoft's MSMQ [12] middleware. They each support their own proprietary messaging APIs and protocols, as well as the standard Java messaging API, the Java Message Service (JMS) [16].

While these products provide a complete and proprietary solution to reliable messaging, a number of open standards for reliable messaging have been emerging. These include the ebXML Message Service [15], the HTTPR protocol [22], the WS-Reliability protocol [9], and the WS-ReliableMessaging protocol [3]. The objective of these specifications is not to promote or prescribe a particular product and middleware infrastructure, but to define the rules governing message delivery, such as for message correlation, persistence, and acknowledgments. HTTPR, for example, can be supported by different messaging agent implementations that are paired with some persistent store; reliable messaging thereby is introduced over the (unreliable) HTTP protocol. WS-Reliability and WS-ReliableMessaging define an XML messaging protocol that allows reliability to be introduced to SOAP messages independent of the underlying transports.

3 Web Services Messaging Using SOAP

The messaging protocol most commonly used for Web services is the Simple Object Access Protocol (SOAP) [5]. SOAP is an XML message format (of an envelope, message headers, and a message body) and a standard encoding mechanism that allows messages to be sent over a variety of transports, including HTTP and JMS.

SOAP-over-HTTP has been the most popular choice for Web services messaging, as it is a well-understood messaging model that is easy to implement and maintain. Due to the frequent use of SOAP-over-HTTP, SOAP is often understood to be a request-response (RPC-like) protocol. However, the synchronous flavor of SOAP-over-HTTP results more from HTTP than from SOAP. If JMS is used as a transport, for example, SOAP can be used to implement asynchronous messaging.

SOAP messaging can take different forms of reliability depending on the underlying transport chosen. While SOAP-over-HTTP is not reliable, SOAP-over-HTTPR ensures that messages are delivered to their specified destination. Similar reliability guarantees can be made for SOAP-over-JMS and SOAP-over-Websphere MQ. On the other hand, a SOAP message itself can be extended to include reliability properties, using the recently proposed WS-Reliability or WS-ReliableMessaging standards. These “extended SOAP” messages then carry relevant reliability information that must be understood and supported by a messaging infrastructure (that

may or may not employ other reliable messaging technology such as Websphere MQ).

It is important to note that SOAP is both a message format and a transport-flexible messaging protocol. With SOAP messaging, we consequently refer to the use of SOAP as both a format and a protocol. To clarify the use of SOAP as a protocol, we refer "SOAP-over-<X>", where <X> is the chosen transport.

SOAP Messaging implies the use of a SOAP library (such as Apache AXIS [1]) to construct, send, receive, and parse SOAP messages, using the desired transport and encoding rules; an application involved in SOAP messaging interfaces the SOAP library, but does not interface any deployed middleware that implements the SOAP transport. If, on the other hand, a middleware like Websphere MQ is used as the messaging protocol and middleware, and SOAP-formatted XML messages are being exchanged (perhaps with the help of some utilities for manipulating SOAP envelopes), we refer to MQ messaging using SOAP/XML messages.

Presently, SOAP as a messaging protocol is not particularly feature rich, making for subtle distinction between SOAP-over-JMS and JMS messaging using SOAP messages (for example.) However, this will change as SOAP matures; the proposed standards for reliable messaging and other specifications addressing, for example, message routing and referral [13] [14] can be seen as evidence for this.

4 Reliable Messaging for Web Services

Reliable messaging for Web services is about achieving reliable messaging aspects (as described in Section 2) within the Web services environment. In this section, we study how existing reliable messaging technologies can be used for this purpose.

A number of reliable messaging technologies exist. These include enterprise message-oriented middleware like IBM Websphere MQ, distributed object messaging standards like JMS, or reliable transport protocols for Web environments like HTTPR. An application may choose to use either one of these technologies, or a combination of these technologies, to address reliable Web services messaging. An application may also, in addition or as an alternative to the above options, choose to implement reliability mechanisms itself as part of the application. Consequently, reliability may (or may not) be addressed on any or all of the protocol/transport, middleware, and application layers.

Figure 1 illustrates the resulting set of common options for implementing reliable messaging in a Web services world:

a) SOAP (with or without a reliability protocol like WS-ReliableMessaging) is used with an unreliable transport (like HTTP); reliability mechanisms are implemented on the application/SOAP messaging layer
b) A reliable transport protocol like HTTPR is used for SOAP messaging (without a reliability protocol like WS-ReliableMessaging); a middleware system (that is, any implementation of messaging agents supporting HTTPR based on HTTP and some persistent storage capability) is required
c) A reliable, proprietary middleware system like IBM Webpshere MQ is used for SOAP messaging (without a reliability protocol like WS-ReliableMessaging); the

middleware defines the transport protocol and provides the necessary distributed infrastructure

d) A reliable messaging standard like JMS is used for SOAP messaging (without a reliability protocol like WS-ReliableMessaging); a JMS implementation is required

e) A reliable, proprietary middleware system like IBM Webpshere MQ (any middleware system leveraged for Web services and with some means of durable storage for message logging qualifies) is directly used, independent of SOAP

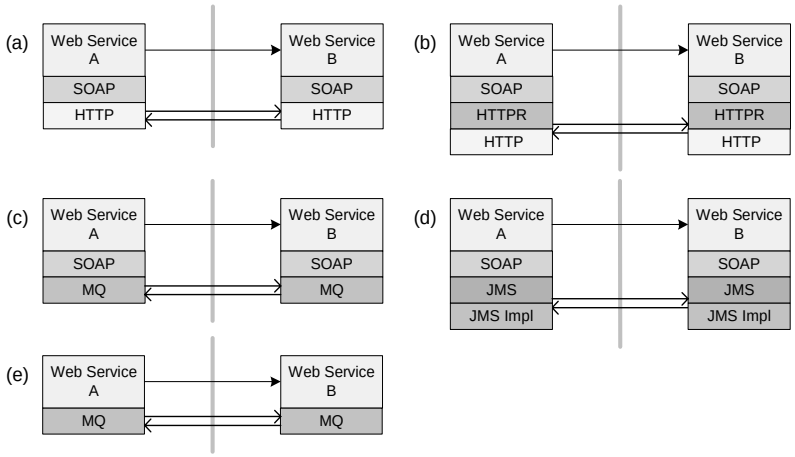

Fig. 1. Reliable Messaging Implementation Options

5 Assessment

As defined in Section 2.2, three facets of reliability exist:

- middleware endpoint-to-endpoint reliability
- application-to-middleware (and middleware-to-application) reliability, and
- application-to-application reliability

In the following, we first examine the above listed options with respect to middleware endpoint-to-endpoint reliability and application-to-middleware reliability. We identify which (required or additional) responsibilities the application developer has when choosing an option. These discussions set the stage to discuss application-to-application reliability.

5.1 Middleware Endpoint-to-Endpoint Reliability

Middleware endpoint-to-endpoint reliability assumes that a middleware is being used for mediation between communicating messaging partners. Middleware endpoint

mediation essentially refers to the idea that messages are made persistent locally on the sender and receiver sides before and after they are being sent (Figure 2). All messages are given unique identifiers, so that a message sender (the endpoint on the sender side) can re-send a message until it gets a positive acknowledgment of receipt by the receiver (the endpoint on the receiver side).

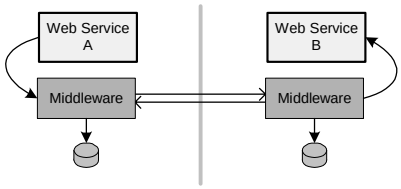

Fig. 2. Middleware Mediation

Option (a). Basic SOAP-over-HTTP does not support endpoint-to-endpoint reliability. HTTP is a synchronous protocol that is "reliable" as long as the connection stays alive (like TCP): it delivers messages at most once, in order, and with a definite acknowledgment for each message delivery or delivery failure. HTTP is unreliable in the sense that when a connection is lost, the message sender will get a connection failure event, but be in doubt about the status of the message:

- The message might not have been delivered to the destination.
- The message might have been delivered to the destination and might have been processed by the destination; the receiver might have replied, or, the receiver might know about the connection failure and may or may not have attempted to rollback its processing, if possible.

To address this uncertainty, the application can mimic middleware mediation on the application layer: The message sender application keeps persistent copies of messages before sending them and attempts message delivery until positive, explicit application acknowledgments confirm the receipt. The receiver application implements the equivalent functionality on its side for reliably sending replies.

If a reliability protocol like WS-Reliability or WS-ReliableMessaging is used, reliability information (for message tracking, for example) is encoded in the SOAP headers. The use of these “standards” allows to ensure endpoint-to-endpoint reliability across distributed services, independent of how the applications choose to implement the reliability mechanisms.

Encoding (lower-level) reliability mechanisms with SOAP messages may however confuse the implementation of higher-level business processes (depending on how the SOAP messages with reliability extensions are created). Furthermore, robust reliable messaging middleware already exist; in cases where a reliable messaging transport for SOAP is used, supporting reliability on the SOAP and on underlying transport layers may cause avoidable inefficiency.

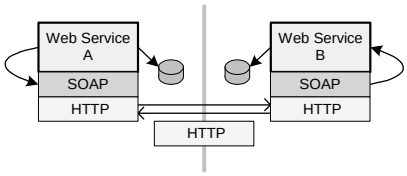

Fig. 3. SOAP-over-HTTP

Option (b). SOAP-over-HTTPR supports middleware endpoint-to-endpoint reliability. The middleware endpoints are the (implementations of) HTTPR messaging agents combined with some persistent storage capability.

An application using this option can either choose to implement the HTTPR messaging agents itself, or select an existing available implementation (such as [11]); additionally, a persistent store at the sender and receiver ends must be installed .

An application sends a SOAP message using a SOAP library (for example, AXIS) that is configured to use an HTTPR sender agent. The SOAP message itself is not HTTPR-specific; the message has the same format as a SOAP-over-HTTP message. The SOAP message is then sent as the body of an HTTPR request; additional required information of message id and correlation information are carried in the HTTPR message context header preceding the HTTPR body. On the receiver side, an HTTPR agent receives the message and directs it further to the SOAP layer. The application's WSDL definitions reflect the use of HTTPR in the bindings specification.

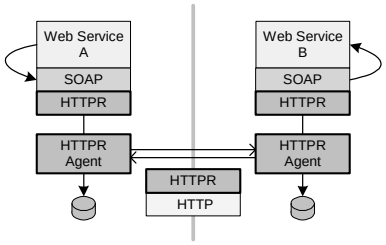

Fig. 4. SOAP-over-HTTPR

Option (c). SOAP-over-MQ also supports middleware endpoint-to-endpoint reliability. The middleware endpoints are message queue managers provided by the messaging middleware product; the persistent stores are message queues.

IBM Websphere MQ is one example of a proprietary message queuing system that supports SOAP messaging. A typical scenario will require the sender and receiver applications to each set up a queue manager (which may be part of a single MQ cluster) and to define a service queue on the receiver side and a response queue on the sender side. The sending application puts a message into the receiver's service queue, and the receiving application consumes the message from the service queue and posts a reply to the sender's queue. A SOAP client library is provided for an application to send SOAP messages over Websphere MQ. On the server side, a SOAP handler processes the incoming messages and invokes the appropriate deployed application.

Note that the message delivery pattern is asynchronous. That is, different from the synchronous HTTP request-response style, MQ (and JMS, see below) request-reply messaging is implemented as two unidirectional, correlated messages (request message and reply message). The sending application is not blocked waiting for the reply, tolerating non-availability of the receiving application.

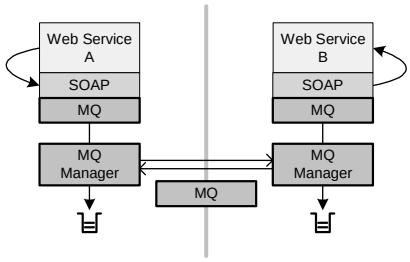

Fig. 5. SOAP-over-MQ

Option (d). SOAP-over-JMS principally compares to SOAP-over-HTTPR and SOAP-over-MQ. Similar to the HTTPR option, a middleware system implementing the JMS standard is required. Any (single) JMS implementation can be used; in case of IBM Websphere MQ, the reliability for SOAP-over-JMS messaging is basically the same as for SOAP-over-MQ.

The middleware endpoints are JMS senders and JMS receivers. Attention must be paid to the different receiver kinds possible: a simple Java client using a JMS MessageListener is not as robust and transactionally reliable as an Enterprise JavaBean (EJB) or a Message-Driven Bean (MDB) (see also Section 5.2).

Using JMS as a transport for SOAP messaging is vendor-proprietary, as (different from SOAP-over-HTTP) no standard has been defined. JMS implementations can also significantly differ from each other (for example, the JMS wire protocol is also vendor-proprietary), so that the SOAP client and the Web services provider must use the same runtime environment to achieve endpoint-to-endpoint reliability (that is, the same vendors' JMS implementation and SOAP-to-JMS transport binding library are required).

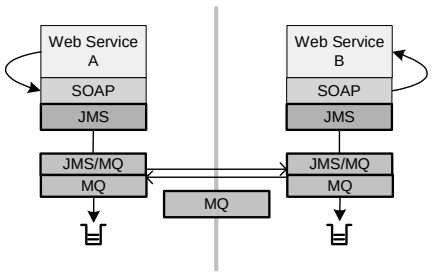

Fig. 6. SOAP-over-JMS

Option (e). Middleware endpoint-to-endpoint reliability can also be achieved by employing enterprise messaging middleware directly, independent of SOAP.

In this case, the middleware must be leveraged to support XML messaging. For example, the middleware may provide an adapter component that takes an XML message (including a standard SOAP message) from a sending application, and then packages the message into its own distributed messaging formats and protocols (such as MQ messages, or JMS messages). The adapter then reliably sends the message to a defined remote queue, using the distributed messaging middleware. On the receiving side, another middleware adapter component reads the message from the queue, unpacks the XML message contained in it, and sends the message to the Web service that is to be invoked.

Benefits of this option include the use of a robust distributed messaging network that may already exist. Furthermore, few changes are required for the sending and receiving application; the invoked Web service may not need to be modified at all.

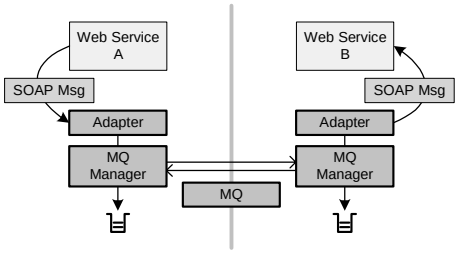

Fig. 7. MQ

5.2 Application-to-Middleware Reliability

Application-to-middleware reliability refers to the reliability features provided by the middleware's application-to-endpoint interface (a.k.a., the messaging API). These may include the following:

- message delivery guarantees (e.g., exactly-once, at-most-once, at-least-once);
- fault-tolerant invocation (of the messaging endpoint);
- the ability to atomically group messaging operations with other application actions.

Depending on the architecture of the messaging middleware, supporting such features poses different challenges. For example, in distributed architectures, where the endpoint is local to the application, fault-tolerant invocation may not be an issue and exactly-once delivery might be supported by the endpoint-to-endpoint reliability mechanisms described above. However, in centralized architectures, where there is a shared messaging server, fault-tolerant invocation becomes more difficult, whereas delivery guarantees become less so. (Hybrid architectures, with centralized messaging servers that are themselves distributed exhibit additional challenges.)

The ability to atomically group messaging operations (e.g., sending a message) with other application activities (e.g., updating a database) may require that the messaging endpoint act as a participant in a two-phase commit protocol (2PC) [2].

For example, if the endpoint is a remote Web service, the WS-Coordination framework [6] and the WS-Transaction "Atomic Transaction" protocols [7] could be used for this purpose; if the endpoint is local, the Java Transaction API [17] or J2EE Connector Architecture [18] could be used.

When assessing the options above with respect to application-to-middleware reliability, we must observe that SOAP itself does not define any such reliability features. Therefore, a SOAP library that supports application-to-middleware reliability features, such as transactionally coordinating messaging operations with other application activities, may have to define proprietary SOAP APIs.

Option (a). When using SOAP-over-HTTP (with or without a reliability protocol for SOAP), the reliability mechanisms may be implemented as part of the application. The application can therefore ensure that its components for creating, storing, and delivering messages are all accessed reliably. For example, to support the atomic grouping of message creation with other application activities, the message store might be implemented as a resource manager that is transactionally coordinated with other resource managers used by the application. Figure 8 illustrates the use of local transactions for this purpose.

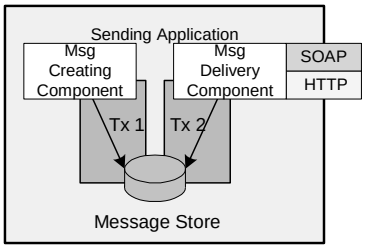

Fig. 8. Local Transactions for Message Storage

In the figure, the transaction that puts a message in the message store (Tx1) typically will further comprise other application activities, thereby grouping the message storage with other activities in an atomic unit-of-work. A second transaction (Tx2) can be defined to read (copy) messages from the message store and to initiate the remote message delivery.

Option (b) to (d). In the cases of SOAP messaging over a reliable transport and middleware (SOAP-over-HTTPR, SOAP-over-MQ, and SOAP-over-JMS/MQ), the underlying middleware implements the reliability mechanisms for endpoint-to-endpoint reliability (including persistent message storage). The application-to-middleware guarantees that can be provided are the message delivery guarantees of the underlying transport and, if the endpoint is local, the reliability of the local procedure calls between the application and the local endpoint. However, unlike Option (a), the application cannot transactionally group a messaging operation with other activities of a larger business process.

Figure 9 illustrates this case. The (sending or receiving) application only interfaces the SOAP library and does not use the underlying middleware directly. The

application uses local calls for SOAP messaging, independent of any functionality that the underlying reliable middleware may offer.

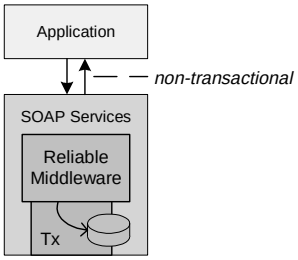

Fig. 9. Non-transactional Use of SOAP Library

If the application requires additional reliability features, in particular the ability to transactionally group messaging operations with other activities as described above, these options are not sufficient. Either the SOAP library would have to expose underlying reliability features (using proprietary extensions) or the application would have to access the underlying middleware directly when these features are needed. However, such solutions may have negative consequences: proprietary extensions will not likely be compatible with future SOAP API standards that do address application-to-middleware reliability, while accessing the underlying middleware directly could interfere with the SOAP libraries use of the middleware.

Option (e). The "direct middleware" option of using JMS or MQ for XML messaging describes the case where the application uses the reliable middleware's API to send and receive messages. Therefore, application-to-middleware reliability relates to the direct use of the underlying middleware's API and its reliability features.

JMS and MQ support, for example, the notion of a "transacted session", allowing a message sender to group a number of messages into an atomic unit-of-work. The middleware then ensures that either all messages of the group are delivered, or none of them. With the use of persistent message queues, support for resource management as required by distributed object transactions is also provided. Therefore, messages put to a message queue can be transactionally coupled with other database and distributed invocations. Figure 10 illustrates this transactional feature.

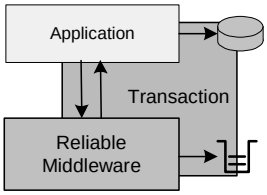

Fig. 10. Transactional Use of Reliable Middleware

5.3 Application-to-Application Reliability

Middleware endpoint-to-endpoint reliability and application-to-middleware reliability provide the foundation on which (higher-level) distributed business processes can be developed. These distributed business processes are constructed from basic application-to-application interactions. Reliability of these basic application-to-application interactions is therefore critical to the reliability of the business process as a whole.

In the following, we briefly discuss ways to support basic application-to-application reliability by traditional means of direct transaction and queued transaction processing [2].

5.3.1 Direct transaction processing

With direct transaction processing, an agreement protocol (e.g., two-phase commit) is used to directly include one application's transaction processing as part of another application's transaction processing. This is shown in Figure 11.

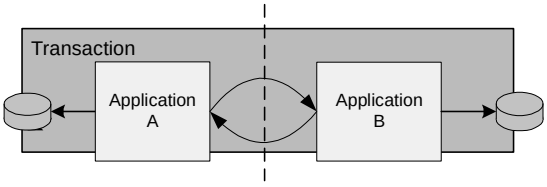

Fig. 11. Direct Transaction Processing

In the figure, Application A and application B interact within the same global transaction. If there is a failure, or if either application decides that the outcome or effect of the interaction is inconsistent, the transaction can be aborted, returning both applications to a consistent state. Furthermore, direct transaction processing can include more than two applications, extending the scope of the reliability guarantee.

The WS-Coordination and WS-Transaction specifications define an interoperable mechanism for supporting direct transaction processing over the Web.

5.3.2 Queued transaction processing

With queued transaction processing, applications interact indirectly using reliable message-oriented middleware; a transaction service, that can integrate messaging resources (e.g., queues) with other resources used by the applications (e.g., databases), is also needed. Employing queued transaction processing for reliable request-reply interactions is shown in Figure 12.

In the figure, three separate transactions are used to ensure that this basic application-to-application interaction is reliable. In transaction 1, application A access local resources and sends a request message to application B; however, the message is not visible to application B until transaction 1 commits. In transaction 2, application B consumes the message, accesses local resources, and sends a response message to application A; the response is not visible to application A until transaction 2 commits.

In transaction 3, application A consumes the response message from application B and accesses local resources. A failure during any single transaction returns the interaction to a well defined, and locally consistent, state.

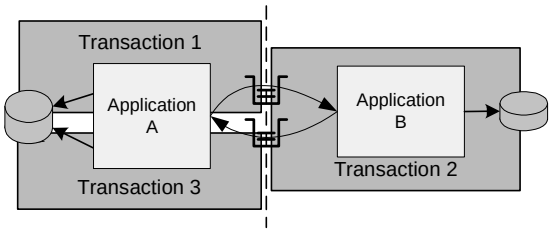

Fig. 12. Queued Transaction Processing

Reliable messaging middleware, supporting application-to-middleware and endpoint-to-endpoint reliability, as described above, can be employed to support queued transaction processing on the web.

5.3.3 Discussion

While direct transaction processing provides a reliable mechanism for multiple applications to agree on the effect and outcome of some joint processing, it can impose tight application coupling, reducing the autonomy the applications involved: for example, 2PC protocols typically require that all participants be active at the same time and can force participants to hold locks (on their local resources) on behalf of other participants.

Yet, queued transaction processing, which supports a looser coupling of applications, can force applications to implement potentially complex logic for coordinating the effects and outcomes of message delivery and processing with the effects and outcomes of other activities (e.g., database updates); if, for example, the response message in Figure 12 indicates that the outcome of application B's processing is inconsistent with application A's processing, application A cannot simply abort its transaction and expect to return to a consistent state (as was the case with direct transaction processing); rather, application A will have to actively correct for the inconsistency, perhaps initiating additional transactions and interactions with application B (for example, a compensating transaction that reverses the work performed in transaction 1).

Our work on conditional messaging [19] and Dependency Spheres [20] propose additional techniques for application-to-application reliability that overcome some of these disadvantages.

6. Summary

In this paper, we presented options for implementing reliable messaging for Web services using existing messaging technology. For each option, we discussed how middleware endpoint-to-endpoint and application-to-middleware reliability is

achieved. We further showed how reliable messaging, when combined with traditional transaction processing techniques (such as direct transaction processing and queued transaction processing), can be applied to support basic application-to-application reliability.

We focused our discussion on the middleware infrastructure for implementing reliable messaging for Web services, as the middleware concern will always be a key challenge independent of which (existing or emerging, future standard) reliable messaging protocol will be used (or if no standard is used). In fact, we believe that any proposed reliable messaging standard must be assessed in consideration of supporting middleware implementations. As we have shown in this paper, there are subtle, but important differences of middleware implementations of Web services that determine to what extent two deployed Web services can exchange messages reliably. This observation may help to estimate implementation efforts and costs related to reliable messaging and therefore, to choose one of the different possible (standards- or non-standards-based) options.

7 References

1. Apache AXIS. http://xml.apache.org/axis/
2. P. A. Bernstein, E. Newcomer. Principles of transaction processing. Morgan Kaufmann, 1997.
3. R. Bilorusets et al. Web Services Reliable Messaging Protocol (WS-ReliableMessaging). BEA, IBM, Microsoft, March 2003. ftp://www6.software.ibm.com/software/developer/library/ws-reliablemessaging.pdf
4. A. Bosworth et al. Web Services Addressing (WS-Adressing). BEA, IBM, Microsoft, March 2003. ftp://www6.software.ibm.com/software/developer/library/ws-addressing.pdf
5. D. Box et al. Simple Object Access Protocol (SOAP) 1.1. W3C Note 08 May 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
6. F. Cabrera et al. Web Services Coordination (WS-Coordination). BEA, IBM, Microsoft, August 2002. http://www-106.ibm.com/developerworks/library/ws-coor/
7. F. Cabrera et al. Web Services Transaction (WS-Transaction). BEA, IBM, Microsoft, August 2002. http://www-106.ibm.com/developerworks/library/ws-transpec/
8. E. Christensen et al. Web Services Description Language (WSDL) 1.1. W3C Note 15 March 2001. http://www.w3.org/TR/wsdl
9. C. Evans et al. Web Services Reliability (WS-Reliability), Version 1.0. Fujitsu, Hitachi, NEC, Oracle, Sonic Software, Sun Microsystems 2003. http://xml.fujitsu.com/en/about/WS-ReliabilityV1.0.pdf
10. IBM Corp. IBM WebSphere MQ. http://www-3.ibm.com/software/ts/mqseries/messaging/
11. IBM Corp. WebSphere MQ Support for Web Services and HTTPR. MA0R Support Pac, IBM Corporation, April 2002. http://www-3.ibm.com/software/ts/mqseries/txppacs/ma0r.html
12. Microsoft. Microsoft Message Queuing (MSMQ). http://www.microsoft.com/msmq/default.htm
13. H. F. Nielsen, S. Thatte. Web Services Routing Protocol (WS-Routing), Microsoft, October 2001. http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-routing.asp
14. H. F. Nielsen et al. Web Services Referral Protocol (WS-Referral), Microsoft, October 2001. http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-referral.asp

15. OASIS. ebXML Message Service Specification Version 2.0. OASIS, April 2002. http://www.oasis-open.org/committees/ebxml-msg/documents/ebMS_v2_0.pdf
16. Sun Microsystems. Java Message Service API Specification v1.1. Sun Microsystems, April 2002. http://java.sun.com/products/jms/
17. Sun Microsystems. Java Transaction API (JTA), Version 1.0.1B. Sun Microsystems, November 2002. http://java.sun.com/products/jta/
18. Sun Microsystems. Java 2 Enterprise Edition: J2EE Connector Architecture Specification, Version 1.0. Sun Microsystems, August 2001 http://java.sun.com/j2ee/connector/
19. S. Tai, T. Mikalsen, I. Rouvellou, S. Sutton. Conditional Messaging: Extending Reliable Messaging with Application Conditions. Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS 2002, Vienna, Austria), IEEE, pp 123-132, July 2002
20. S. Tai, T. Mikalsen, I. Rouvellou, S. Sutton. Dependency-Spheres: A Global Transaction Context for Distributed Objects and Messages. Proceedings of the 5th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2001, Seattle, USA), IEEE, pp 105-115, September 2001
21. TIBCO. TIBCO Rendezvous. http://www.tibco.com/solutions/products/active_enterprise/rv/default.jsp
22. S. Todd, F. Parr, M. Conner. A Primer for HTTPR. An Overview of the Reliable HTTP Protocol. IBM Corporation, July 2001. http://www-106.ibm.com/developerworks/webservices/library/ws-phtt/