An Analysis of Reliable Delivery Specifications for Web Services

Shrideep Pallickara, Geoffrey Fox and Sangmi Lee
(spallick, gcf, leesangm)@indiana.edu
Community Grids Laboratory, Indiana University

Abstract

Reliable delivery of messages is now a key component of the Web Services roadmap, with two promising, and competing, specifications in this area viz. WS-Reliability from OASIS and WS-ReliableMessaging from IBM and Microsoft. In this paper we provide an analysis of these specifications. Our investigations have been aimed at identifying the similarities and divergence in philosophies of these specifications. We also include a gap analysis and a series of recommendations plugging the gaps identified by the gap analysis.

Keywords: guaranteed delivery, ordered delivery, WS-Reliability, WS-ReliableMessaging and fault tolerance.

1. Introduction

Remote method invocations have been used in distributed systems for quite some time. Frameworks such as CORBA from the Object Management Group (OMG) [1] have had schemes in place to facilitate invocations on remote objects for more than a decade. There also has been support for remote invocations in programming languages, a case in point being the Java Remote Method Invocation (RMI) Framework [2]. In these cases we could think of the remote object as providing a service comprising a set of functions. The provider exposes the service's capability through an appropriate description language, which comprises the function names, the number and type arguments that a given service function takes and finally the return type that would be returned upon completion of the invocation.

The underlying principle for Web Services [3] is similar to what existed in these earlier systems. The difference lies in the scale, scope, ubiquity and easy of utilization of these services. The deployments and utilization of these services are driven by a slew of XML based specifications that pertain to exposing services, discovering them and accessing these securely once the requestor is authenticated and authorized.

As web services have matured the interactions that the services have between themselves have gotten increasingly complex and sophisticated. Web services can be composed easily from other services, and these services can be made to orchestrate with each other in dynamic fashion. Web services specifications have addressed issues such as security, trust, notifications, service descriptions and invocations among others. These specifications can leverage, extend and interoperate with other specifications to facilitate incremental addition of features and capabilities. As web services have become dominant in the Internet and Grid systems landscape, a need to ensure guaranteed delivery of interactions (encapsulated in messages) between services has become increasingly important. This highly important and complex area was previously being addressed in the Web Services community using homegrown proprietary application specific solutions. It should be noted that the terms guaranteed delivery and reliable delivery tend to be used interchangeably to signify the same concept.

Reliable delivery of messages is now a key component of the Web Services roadmap, with two promising, and competing, specifications in this area viz. WS-Reliability from OASIS [4] and WS-ReliableMessaging [5] from IBM and Microsoft. In this paper we provide an analysis of these specifications. Our investigations have been aimed at identifying the similarities and divergence in philosophies of these specifications. We also include a gap analysis identifying potential drawbacks in both these specifications, including a series of recommendations to issues identified in the gap analysis. We believe it is inevitable that these specifications may continue to exist alongside each other. To account for such a scenario we also include a scheme for federating between these specifications. Such a scheme will allow service nodes to belong to either one of these competing specifications and still continue to interact reliably with each other.

This paper is organized as follows in section 2 we provide an overview of the related work in this area. In section 3 we include a brief primer on acknowledgements the most fundamental element in ensuring guaranteed delivery. In sections 4 and 5 we provide an analysis of the similarities and differences in these specifications. In section 6 we present a gap analysis of these specifications, while providing recommendations to plug these gaps in a series of recommendations provided in section 7.

2. Related work

In this section we provide a taxonomy of related work in the area of reliable and ordered delivery. We consider traditional group based systems, asynchronous publish/subscribe systems, message queuing systems,

distributed object based systems and finally recovery oriented computing. The efforts in group based systems and publish/subscribe systems have focused on ensuring reliable delivery to multiple entities interested in a message. Messaging queuing systems deal with ensuring reliable delivery between queues. Fault tolerant CORBA tries to ensure availability (and accompanying accesses) of the remote object in question under various failure scenarios.

2.1. Group based systems

The problem of reliable delivery [6, 7] and ordering [8, 9] in traditional group based systems with process crashes has been extensively studied. The approaches normally have employed the primary partition model [10], which allows the system to partition under the assumption that there would be a unique partition which could make decisions on behalf of the system as a whole, without risk of contradictions arising in the other partitions and also during partition mergers. However the delivery requirements are met only within the primary partition. Recipients that are slow or temporarily disconnected may be treated as if they had left the group.

This virtual synchrony model, adopted in Isis [11], works well for problems such as propagating updates to replicated sites. Systems such as Horus [12] and Transis [13] manage minority partitions (by having variants of the virtual synchrony model) and can handle concurrent views in different partitions. The overheads to guarantee consistency in these cases can be too strong

Spinglass [14] employs “gossip” style algorithms, where recipients periodically compare their disseminations with other members of the group. Each recipient compares it dissemination sequence (a message digest of the message sequences received so far) with one of the group members. Deviations in the digest result in solicitation requests (or unsolicited responses) for missing messages between these recipients. This approach is however unsuitable where memberships can be fluid and hence a recipient is unaware of other recipients that should have received the same message sequences. Approaches to building fault-tolerant services using the state machine approach have been suggested in Ref [15].

2.2. Publish/Subscribe systems

NaradaBrokering [16, 17] facilitates delivery of events to interested entities in the presence of node and link failures. Furthermore, entities are able to retrieve any events that were issued during an entity’s absence (either due to failures or an intentional disconnect). The scheme withstands failures of the entire broker network and does not require a stable storage at every entity.

DACE [18] introduces a failure model, for the strongly decoupled nature of pub/sub systems. This model tolerates crash failures and partitioning, while not relying on consistent views being shared by the members. DACE achieves its goal through a self-stabilizing exchange of views through the Topic Membership protocol. This however may prove to be very expensive if the number and rate at which the members change their membership is high.

The Gryphon [19] system uses knowledge and curiosity streams to determine gaps in intended delivery sequences. This scheme requires persistent storage at every publishing site and meets the delivery guarantees as long as the intended recipient stays connected in the presence of intermediate broker and link failures. It is not clear how this scheme will perform when most entities within the system are both publisher and subscribers, thus entailing stable storage at every node in the broker network. Furthermore it is conceivable that the entity itself may fail, the approach does not clearly outline how it handles these cases. Systems such as Sienna [20] and Elvin [21] focus on efficiently disseminating events, and do not sufficiently address the reliable delivery problem in the presence of failures.

2.3. Message Queuing Systems

Message queuing products (MQSeries) [22] are statically pre-configured to forward messages from one queue to another. This leads to the situation where they generally do not handle changes to the network (node/link failures) very well. Furthermore these systems incur high latency since they use the store-and-forward approach, where a message is stored at every stage before being propagated to the next one. They also require these queues to recover within a finite amount of time to resume operations.

2.4. Fault Tolerant CORBA

The Fault Tolerant CORBA (FT-CORBA) [23] specification from the OMG defines interfaces, policies and services that increase reliability and dependability in CORBA applications. The fault tolerance scheme used in FT-CORBA is based on entity redundancy [24], specifically the replication of CORBA objects. In CORBA objects are uniquely identified by their interoperable object reference (IOR). The FT-CORBA specification introduces interoperable group object references (IGOR). When a remote object, the client can access a replica simply by iterating through the references contained in the IGOR until the invocation is successfully handled by the replicated object. The specification introduces several schemes to manage different replication schemes.

The DOORS (Distributed Object-Oriented Reliable Service) system [25] incorporates strategies to augment implementations of FT-CORBA with real time characteristics. Among the issues that the DOORS system

tries to address are avoiding expensive replication strategies and dealing with partial failure scenarios. DOORS provides fault tolerance for CORBA ORBs based on the service approach. Approaches such as Eternal [26] and Aqua [27], provide fault tolerance by modifying the ORB. OS level interceptions of have also been used to tolerate faults in applications. Ref [28] provides an excellent taxonomy of the various approaches to fault tolerant CORBA.

2.5. Recovery Oriented Computing

The Recovery Oriented Computing (ROC) project [29] at UC Berkley and Stanford University takes the perspective that faults, failures, errors and bugs are facts to be coped with rather than problems to be solved (also known as Peres' law). The project is deals with reducing the Mean Time To Recover (MTTR) from system failures instead of Mean Time To Failure (MTTF). ROC improves dependability in systems by recovering from failures fast thus ensuring continued availability.

3. A primer on acknowledgements

Entities involved in reliable messaging need to facilitate easy detection of errors in received sequences while also being able to fix these errors in sequences. In sender-initiated protocols a sender get positive acknowledgments from all receivers periodically. A positive acknowledgement (ACK) confirms the receipt of a specific event by a given receiver. This information along with the knowledge of the events, which an entity is supposed to receive, allows the identification of holes in the delivery sequence at any given node. The sender can then initiate retransmissions to fix these errors.

In receiver-initiated protocols errors in received sequences are detected at the receivers, This detection in turn triggers negative acknowledgements (NAK) to fix these holes in the delivered sequences and retrieve any previously undelivered events. Some schemes rely only on one of these acknowledgement mechanisms. In receiver initiated protocols the assumption at the sender is that the message has been received at the receiver unless indicated otherwise by the NAKs.

It should be noted that in sender-initiated protocols the error detection, initiation of error correction and the retransmission are all performed at the sender side. In receiver-initiated protocols the error detection and initiation of error corrections are performed at the receiver, while the retransmissions are performed by the sender. ACK based schemes can exist by themselves, while NAK based schemes cannot. This is because in a purely NAK based scheme there is no way for the sender to know for sure if a message was received and hence the sender can never clear the buffer allocated for messages that were sent by the sender.

4. Similarities in the specifications

The specifications, both of which are based on XML, address the issue of ensuring reliable delivery between two service endpoints. In this section we outline the similarities in the underlying principles that guide both these specifications. The similarities that we have identified are along the six related dimensions of acknowledgements, ordering and duplicate eliminations, groups of messages and quality of service, timers, security and fault/diagnostic reporting.

Both the specifications rely only on positive acknowledgements to ensure reliable delivery. This in turn implies that all error detections, initiation of error corrections and subsequent retransmissions of "missed" messages are performed at the sender side. The receiver side plays no role whatsoever in detecting these errors and initiating corrections. A sender may also proactively initiate corrections based on the non-receipt of acknowledgements within a pre-defined interval.

The specifications also address the related issues of ordering and duplicate detection of messages issued by a source. A combination of these issues can also be used to facilitate exactly once delivery. Both the specifications facilitate the guaranteed exactly-once delivery of messages, a very important quality of service that is highly relevant for transaction oriented applications specifically banking, retailing and e-commerce.

Both the specifications also introduce the concept of a group of messages. All messages that are part of a group of messages share a common group identifier. The specifications explicitly incorporate support for this concept by including the group identifier in protocol exchanges that take place between the two entities involved in reliable communications. Furthermore, in both the specifications the qualities of service constraints that can be specified on the delivery of messages are valid only within a group of messages, each with its own group identifier.

The specifications also introduce timer based operations to both messages and groups of messages. Individual and groups of messages are considered invalid upon the expiry of timers associated with them. Finally, the delivery protocols in the specifications also incorporate use timers to initiate retransmissions and to time out retransmission attempts.

In terms of security both the specifications aim to leverage the WS-Security specification which facilitates message level delivery. Message level security is independent of the security of the underlying transport and facilitates secure interactions over insecure communication links.

The specifications also provide for notification and exchange of errors in processing between the endpoints involved in reliable delivery. The range of errors supported in these specifications can vary from an inability to decipher a message's content to complex errors pertaining to violations in implied agreements between the interacting entities.

5. Divergence in philosophies

In this section we compare the divergence in the philosophies of some key concepts in these specifications.

5.1. SOAP related issues

WS-ReliableMessaging specifies an XML based schema for elements that are needed for reliable messaging. WS-ReliableMessaging includes a SOAP binding for its protocol. WS-Reliability, on the other hand, is a SOAP-based protocol for the reliable delivery of messages. WS-Reliability includes a HTTP binding, where the HTTP response can be used for carrying acknowledgements associated with individual messages. Similarly, SOAP faults as a result of processing and protocol errors can also be carried during these HTTP responses.

5.2. Grouping Messages

In WS-ReliableMessaging every message is considered to be part of a group of messages. Even if there is just a single message, it is considered to be part of a sequence comprising only one message. The unique identifier associated with a Message comprises the unique Group identifier and its position within the group of messages.

WS-Reliability on the other hand allows a message to exist outside the realms of a group of messages. Every message also has its own unique identifier.

5.2.1 Beginning sequences

Messages within a group of messages are identified differently in both the specifications. In WS-Reliablity when a sender node is ready to start issuing a group of messages, the exchange between the sender and receiver also includes a `SequenceNumber` element with a status attribute. The `status` attribute can take one of three values – `start` indicating the beginning; `end` indicating the last message and `continue` for every message that is neither the first or the last message within the sequence.

In WS-ReliableMessaging the last message in the group of message is identified through the use of the `LastMessage` element in the Sequence element. There is a `MessageNumber` associated with every message in a group of messages, and this is what is used to identify the beginning and position of messages in a group of messages. The absence of the `LastMessage` element is indicative of the fact that the end of the sequence of messages has not yet been reached. Furthermore, a receiver node is issues a fault if any other message in the same group of messages has a `MessageNumber` greater than the one contained in the message with `LastMessage` element.

To ensure the consistency of processing messages belonging to different groups of messages, it is important to make sure that no messages are issued in a group of messages once a message has been explicitly tagged as the last message in the sequence. This important verification does not currently explicitly exist in the WS-Reliability specification.

5.3. Sequence Numbering

Every message within a group of messages in both these specifications has numbering information associated with it (`SequenceNumber` in WS-Reliability and `MessageNumber` in WS-ReliableMessaging). The numbering begins at 0 for WS-Reliability while it begins at 1 for WS-ReliableMessaging.

From the implementation standpoint of these specifications, the following issue needs to be considered. In most languages the default values associated with variables that are not explicitly initialized is 0. In an implementation of the WS-ReliableMessaging specification this value needs to be explicitly updated (since a value of zero is a invalid number) prior to routing it to the receiver. Thus, there is no ambiguity regarding whether the variable associated with the sequence numbering was initialized or not. This is not the case with WS-Reliability where it is conceivable that a message can be published without incrementing the value assigned by default. In general it is preferable that the sequence number is explicitly increased prior to issuing the message instead of doing so after sending the message across.

Within a group of messages WS-Reliability does not require the numbering information to be present in every message. In fact the numbering information is mandated only for ensuring ordered delivery. In the WS-ReliableMessaging case the numbering information is necessary for every message. This is also because, unlinke WS-Reliability, messages in WS-ReliableMessaging do not have a separate unique identifier associated with them. The unique identifier associated with the message is a combination of the group identifier and the message number.

In the unlikely event that there is a rollover associated with messaging numbering, both these specifications handle the issue in different ways. In WS-Reliability case the source is expected to generate a new group identifier and begin new sequence only after receipt of the last

message in the older message sequence. In the WS-ReliableMessaging case no new sequences can be generated and a `MessageRollover` fault needs to be issued.

5.4. Acknowledgements

Both the approaches rely on positive acknowledgements. Error detection and the accompanying corrections are initiated by the sender upon the receipt of acknowledgements from the receiver. There are some differences in the acknowledgement scheme in these specifications. In the WS-ReliableMessaging case, the receiver need not acknowledge the receipt of every message. When the messages being sent are part of a sequence, the last message in the sequence has an indicator indicating that it is indeed the last message. Upon receipt of this message, the receiver sends a report regarding the messages that it received. This includes, besides the identifier associated with the message group, the range of the messages that was received and also individual sequences numbers in the case there were missing sequences between these messages. In WS-ReliableMessaging an entity can also explicitly specify, using the `AckRequested` element, that an acknowledgement is mandated upon the receipt of any message. We recommend that this be mandatory for the last message in a sequence of messages.

The WS-Reliability specification on the other hand mandates an acknowledgement for *every* message that is received at the receiver. The acknowledgements are based on the message ID and not based on specific sequence numbers. As mentioned earlier, the sequence numbers are used only for satisfying the ordering constraint. In WS-Reliability the sender can also specify the reply pattern for a receiver of its messages. The acknowledgements may then be issued directly in the reply to a reliable message, as a callback request and finally or in the response to a separate poll request.

The approach deployed by the WS-Reliability specification facilitates earlier detection of lost messages and concomitant retransmissions to heal these errors. On the other hand, in the default mode in WS-ReliableMessaging the delays associated with error detection may be significantly higher if the number of messages present in a group of messages is high.

5.5. Ordering and duplicate detection

Duplicate detections and ordering are performed at the receiver side, which needs to decide the rejection of message and the ordering associated with messages. These guarantees are valid only within a sequence of messages, and since the numbering information associated with the messages is known to increase monotonically, ordering can easily be ascertained. In WS-ReliableMessaging both ordering and duplicate detection is based on the numbering information associated with individual messages within a sequence. The duplicate detection can however exist independent of ordered delivery.

In WS-Reliability the numbering information associated with messages comes into play only while ensuring ordered delivery. Duplicate detection of messages is based on the message identifiers associated with individual messages. This can sometimes prove to be an expensive operation based on the number of messages in the sequence.It should be noted that in WS-Reliability Order is always tied to guaranteed delivery and cannot be separately specified.

5.6. Quality of Service

In WS-Reliability the quality of service associated with the delivery of messages is dictated entirely by the source of messages. In WS-ReliableMessaging this information can be specified by the receiver side through the use of the WS-Policy family of specification to arrive on assurances associated with the delivery of messages. WS-Policy enables a receiver to describe and advertise its capabilities and/or requirements, and enables communication regarding the selected characteristics that apply for a given sequence of messages. The delivery assurances availble in WS-ReliableMessaging include AtmostOnce, AtleastOnce, ExactlyOnce and InOrder.

Since the specification pertains to reliable delivery the AtmostOnce delivery assurance can imply non-delivery of a given message and should not be there in the first place.

5.7. Timestamps and expiry related information

In WS-Reliability the timestamps are based on UTC timestamps and conform to the dateTime element specified in [XML Schema Part2: Data Types]. In WS-ReliableMessaging there is no explicit reference to UTC though the timestamps use the `dateTime` format alluded to earlier.

Both the specifications provide for timer based expiry pertaining to an individual message or groups of messages. In WS-Reliability there is an `ExpiryTime`, which defines the expiration time associated with an individual message. The specification also includes `removerAfter` where the receiver maintains the group identifier until the end of the sequence of messages is received or after the expiry of the specified time.

In WS-ReliableMessaging `Expires` provides an indication of the expiry time for a sequence of messages. The specification also incorporates an **`Inactivity`** timeout (specified in milliseconds) which is the duration after which an endpoint that has not received application or control messages belonging to a sequence of messages

may consider the aforementioned sequence to have been terminated due to inactivity.

5.8. Retransmissions

Retransmissions are always initiated by the sender side, this is an artifact of the use of positive acknowledgements. Retransmissions can also be proactive where multiple successive attempts at varying intervals (usually increasing) are made to deliver a message. In WS-Reliability the retransmissions are triggered faster since an acknowledgement is expected corresponding to the delivery of every message. The source attempts retransmissions until a specified number of attempts have been made.

WS-ReliableMessaging on the other hand allows the specification of a `RetransmissionInterval` (specified in milliseconds) for a sequence of messages. This in turn affects every message within that sequence of messages and may be modified at the source's discretion during the lifetime of the sequence. The specification also allows this interval to be adjusted based on the exponential backoff algorithm. In WS-ReliableMessaging the efficiency of error corrections is determined based on the acknowledgement policy (one per message, or after the delivery of a sequence) and the time specified in the `RetransmissionInterval.`

5.9. Fault Codes supported within the protocol

The faults reported by the protocols facilitate error reporting based on the problems that can be reported. WS-Reliablity facilitates the reporting of invalid message headers, invalid message identifiers and invalid references to message identifiers in acknowledgements issued by a receiver. The specification also facilitates the reporting of invalid timestamps and expiry times associated with messages. Finally, the specification can also report errors in reply patterns.

The WS-ReliableMessaging specification can report errors pertaining sequence terminations, message number rollovers, invalid acknowledgements, invalid sequences and errors pertaining to the maximum numbering information that can be associated with messages in a given sequence.

While both specifications sufficiently address faults/errors that can take place WS-Reliability lacks the ability to report violation of the maximum numbering information that can be associated with messages in a given sequence. WS-ReliableMessaging on the other hand lacks the ability to notify errors pertaining to the timing related operations within the specifications.

6. Gap Analysis

In this section we present a gap analysis of the two specifications. This analysis pertains to what we feel is lacking in both these specifications. The gap analysis is performed over specific items, each of which is described in individual subsections.

6.1. Sender side error detections

Error detections are performed only at the sender side. This happens to be the case even though the receiver side possesses information to perform this. Since error detections are performed at the sender side, the receiver side can never initiate the accompanying error corrections. This in turn implies that latencies for delivery of events is bound by the retransmission interval setup at the sender side.

6.2. Sender side error corrections

Since the retransmissions are always initiated at the sender side, the error corrections are bound both by the retransmission interval and the number of attempts that will be made before the retransmission is timed out. It is conceivable that an entity might have disconnected (either due to failures, scheduled reboots or application startups). In this case proactively initiating retransmissions tend to be futile and a waste of system and network resources. The costs increase with the number of attempts, unacknowledged messages and payload sizes of these messages.

It is also possible that the application logic processing a certain type of message could be the cause for the problem, in which case the receiving entity might be forced to defer the acknowledgement until the problem has been rectified. In this case constant retransmissions from the sender side might need to be continually discarded.

6.3. Support for mobile computing

An area of increasing significance, given the proliferation of sensors and the advances in wireless networks, is mobile computing. Of specific concern within mobile computing is the premium on network usage with limited and hence slower computation speeds.

Proactive retransmissions that can sometimes flood a receiver node (as outlined earlier) can exacerbate problems on a mobile device. In such cases the inability in both these specification to facilitate receiver-initiated error detections and subsequent retransmissions can seriously impact the performance of these devices.

6.4. Support for ordered delivery across sequences

Both the specifications do not provide for ordered reliable delivery across sequences. The specifications do not facilitate the setting up of causal relationships between messages published in different groups.

This can be a problem since the sender might not know ahead of time the causal/general ordering relationships that will exist between messages that it might publish. It is thus impossible to ensure that a message would be delivered *after* the delivery of a previously published message belonging to a different group.

6.5. Support for one-to-many reliable & ordered delivery

The specifications also implicitly assume that a need would never arise to ensure ordered delivery across multiple services. It is conceivable that a service may need to ensure that a given message was received by some other service before it publishes a message to some other service B. We argue that the problem is sufficiently complex and important enough that it should have been addressed within this specification.

6.6. Complexity of duplicate detections

Both these specifications facilitate duplicate detection only within a group of messages. In WS-ReliableMessaging this is based on the sequence number associated with messages and not on the identifier that might be associated with the message. In fact messages do not have a separate identifier and the identifier is determined based on the message number and the identifier associated with the sequence.

If a group of messages does not have an expiration time based on the current scheme, in both these cases, the receiver might need to maintain information indefinitely regarding the sequences that have been delivered. The specifications do not describe the effect of the same message being present in different sequences. The problem is compounded in the WS-ReliableMessaging specification where a message does not have a unique identifier associated with it. It is thus possible for the same message to be assigned different numbering information with the same sequence!

In WS-Reliability duplicate detection is based on message identifiers. In such cases the complexity of duplicate detections increases with number of messages in a sequence. In fact the problem becomes combinatorially explosive as the number of messages in the group of messages approaches the message rollover limit.

6.7. Trivial implementations of the specification

Both the specifications provide for the expiry of messages and groups of messages. It is obvious that the delivery constraints associated with these messages would no longer be valid since the message itself is an invalid message. Introducing expiry timers associated with messages and sequences, provides an escape clause, which in turn to could be used to have trivial implementations of the specification. If all messages have a timer element associated with them, an trivial implementation may simply discard all messages that it is expected to route reliably.

Expiry of messages and sequences are orthogonal to the reliable delivery guarantee, and should be handled by the application layers.

7. Recommendations

In this section we provide recommendations to plug the gaps that were identified in the previous section.

7.1. Use Negative acknowledgements

We recommend the use of Negative acknowledgements (NAKs). NAKs enable the receiver side to detect errors in received sequences and to initiate retransmissions. Such a scheme ensures that receiver is ready to process the retransmissions and that the chances of the retransmissions not being processed at the receiver side are lower.

Such a scheme can also ensure that there are two entities that can initiate retransmissions and error corrections. The onus is otherwise only on the sender side. This scheme allows proactive sender retransmissions and detection of missing sequences based on the received acknowledgements to exist alongside the scheme we recommend.

The NAK scheme would also assuage the strains imposed on mobile devices that may use these protocols to ensure reliable delivery.

7.2. Every message should have a catenation number

To ensure ordered delivery across sequences, we recommend that every message should have a monotonically increasing catenation number associated with it. Furthermore, we recommend that this catenation numbering information associated with a message identifier should have a one-to-one relationship i.e. for a given message identifier there is only one numbering information and vice versa. This catenation numbering information allows a receiver to order all messages issued by a source irrespective of the sequence, which they belong to.

To support cases where a rollover might occur at a destination, we also include rollover epoch signifying the rollover horizon that the event belongs to. By having the catenation and rollover epochs both as Unsigned Long variables, this scheme allows a source to uniquely identify 18,446,744,073,709,551,615 x 18,446,744,073,709,551,615 messages that it can issue.

7.3. Including previous catenation numbers with every messages

To facilitate ordering/delivery-constraints for messages across multiple receiver destinations, for every message, belonging to a specific group of messages, the source also needs to also add information regarding the catenation number associated with the last message published to that destination. A sender node can then wait until it receives an acknowledgement from a receiver prior to issuing cross-destination-dependent message to another receiver, which is supposed to receive this event only after the acknowledgement. The source can of course issue messages to receivers at other destinations if the source decides to do so.

Including previous catenation numbers also facilitates ordering of messages across sequences and receiver destinations. Thus, for two groups of messages A and B, for totally ordered delivery across sequences of messages at a destination, a message should be delivered only if the message corresponding to the previous catenation number was previous delivered reliably and in order. This can be a recursive constraint, meaning that a previous sequence number should be delivered reliably and in order.

To facilitate ordered delivery within sequences, a message could also include information regarding the catenation number that was associated with the last message published in that sequence (a value of zero would signify that the message in question is the first one within that sequence).

7.4. Easier duplicate detections

Since every message, issued by a source would have a unique catenation number associated with, duplicate detections are easier to perform. Specification and evaluation of causal constraints based on this catenation number are also easier to perform.

7.5. Elimination of time based expiry in messages and sequences

We recommend the removal of timer based expiry of messages and sequences, which we believe are application specific and clearly outside the realms of the reliable delivery specifications. Such a move would eliminate trivial implementations and result in a specification that truly provides a basis for reliable delivery.

8. Federation between these specifications

We believe that it is possible that these specifications might be deployed concurrently. Federation between these specifications will allow endpoints in these specifications to interact with each other. This would involve mapping the semantics of operations involved in these specifications. These operations need to be managed by a middleware. In this section we identify the key issues that need to be considered while federating between these specifications in each of the two cases that need to be considered. The quality of service that can be negotiated between the sender and receiver is based on the strongest constraint that is available between these entities. This would imply that in both the cases, the quality of service available would be reliable delivery and exactly-once-ordered-reliable delivery.

8.1. WS-Reliability Sender and WS-ReliableMessaging Receiver

In this case, any standalone messages delivered at the receiver need to be part of a sequence of messages. The middleware is thus responsible for packing the standalone message such that it is part of a group of messages with message number 1. The generated group identifier could be the same as the message identifier in the original message.

Acknowledgements will be mandated for every message issued by the WS-Reliability Sender. The acknowledgements issued by the WS-ReliableMessaging node will be based on group identifiers and Message Number. While this would be straightforward for standalone messages, in the case of groups of messages, the middleware is expected to keep track of the identifiers associated with the individual message numbers and issue the acknowledgement (based only on the message identifier) to the WS-Reliability nodes.

The numbering of messages in the sequence also needs to be addressed. The middleware is expected to increment the numbering information associated with individual messages in a sequence by one. This is to account for the fact that messages are numbered starting at 1 in WS-ReliableMessaging and 0 for WS-Reliability. The WS-Reliability node should initiate the creation of a new group of messages when it publishes a message with numbering information that is just once increment away from the maximum value of an Unsigned Long. This is to prevent the occurrence of a MessageRollover fault in WS-ReliableMessaging receivers.

The expiry timer associated with an independent message in WS-Reliability can be mapped into the expiry

timer associated with the corresponding WS-ReliableMessaging sequence with only one message. Mapping corresponding to expiry of sequences can be easily mapped.

8.2. WS-ReliableMessaging Sender and WS-Reliability Receiver

In this case every message issued is part of a sequence of messages. The acknowledgement that the sender expects is based on the group identifier and the numbering information associated with the message. This mapping of message identifier to message number within a sequence of messages needs to be performed by the middleware.

Also, message sequence numbers should be decremented prior to delivery at the receiver node. This is because the WS-Reliability node expects the message numbering to start at 0. As far as Message numbering rollovers are concerned, the middleware is expected to throw a `MessageRollover` fault as soon as it encounters a message numbering rollover.

The mapping of expiry timers is easy in this case too.

9. Conclusions

The specifications pertaining to the reliable delivery of messages is one of the most important set of specifications in the Web Service landscape. In this paper we have presented an analysis of the two most promising and leading specifications specified by traditional leaders in Web Service efforts. Some of our comparisons pertaining to WS-Reliability and WS-ReliableMessaging have been summarized in the form of table in Table 1 on page 10.

10. References

[1] The Object Management Group http://www.omg.org

[2] The Java Remote Method Invocation Framework http://java.sun.com/products/jdk/rmi/

[3] The W3C Web Services Acivity. http://www.w3c.org

[4] Web Services Reliable Messaging TC WS-Reliability. http://www.oasis-open.org/committees/download.php/5155/WS-Reliability-2004-01-26.pdf

[5] Web Services Reliable Messaging Protocol (WS-ReliableMessaging) http://www-106.ibm.com/developerworks/webservices/library/ws-rm/

[6] Kenneth Birman. The process group approach to reliable distributed computing. Communications of the ACM, 36(12):36–53, 1993.

[7] Vassos Hadzilacos and Sam Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR94-1425, Dept. Of Computer Science, Cornell University, Ithaca, NY-14853, May 1994.

[8] Kenneth Birman. A response to Cheriton and Skeen's criticism of causal and totally ordered communication. Technical Report TR 93-1390, Dept. Of Computer Science, Cornell University, Ithaca, NY 14853, October 1993.

[9] Kenneth Birman and Keith Marzullo. The role of order in distributed programs. Technical Report TR 89-1001, Dept. Of Computer Science, Cornell University, Ithaca, NY 14853, May 1989.

[10] Aleta Ricciardi, Andre Schiper, and Kenneth Birman. Understanding partitions and the "no partition" assumption. In Proceedings of the Fourth Workshop on Future Trends of Distributed Systems, Lisbon, Portugal, September 1993.

[11] Kenneth Birman. Replication and Fault tolerance in the ISIS system. In Proceedings of the10th ACM Symposium on Operating Systems Principles, pages 79–86, Orcas Island, WA USA, 1985.

[12] R Renesse, K Birman, and S Maffeis. Horus: A flexible group communication system. In Communications of the ACM, volume 39(4). April 1996.

[13] D Dolev and D Malki. The Transis approach to high-availability cluster communication. In Communications of the ACM, volume 39(4). April 1996.

[14] Spinglass: Secure and Scalable Communications Tools for Mission-Critical Computing. Kenneth P. Birman, Robbert van Renesse and Werner Vogels. International Survivability Conference and Exposition. DARPA DISCEX-2001, Anaheim, California, June 2001.

[15] Fred Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. In ACM Computing Surveys, volume 22(4), pages 299–319. ACM, December 1990.

[16] The NaradaBrokering System http://www.naradabrokering.org

[17] Shrideep Pallickara and Geoffrey Fox. NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. Proceedings of ACM/IFIP/USENIX International Middleware Conference. 2003.

[18] Romain Boichat, Patrick Th. Eugster, Rachid Guerraoui, and Joe Sventek. Effective Multicastprogramming in Large Scale Distributed Systems. Concurrency: Practice and Experience, 2000.

[19] Sumeer Bhola, Robert E. Strom, Saurabh Bagchi, Yuanyuan Zhao, Joshua S. Auerbach: Exactly-once Delivery in a Content-based Publish-Subscribe System. DSN 2002: 7-16

[20] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In Proceedings of the 19th ACM Symposium on Principles of Distributed Computing, pages 219–227, Portland OR, USA, July 2000.

[21] Bill Segall and David Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In Proceedings AUUG97, pages 243–255, Canberra, Australia, September 1997.

[22] The IBM WebSphere MQ Family. http://www-3.ibm.com/software/integration/mqfamily/

[23] Object Management Group, Fault Tolerant CORBA Specification. OMG Document orbos/99-12-08 edition, December 1999.

[24] Object Management Group, Fault Tolerant CORBA Using Entity Redundancy RFP. OMG Document orbos/98-04-01 edition, April 1998.

[25] Balachandran Natarajan, Aniruddha Gokhale, Douglas Schmidt and Shalini Yajnik. "DOORS: Towards High-performance Fault-Tolerant CORBA", in Proceedings of the

2nd International Symposium on Distributed Objects and Applications (DOA), Antwerp, Belgium, Sept 2000.
[26] Priya Narasimhan, Louise E. Moser, P. M. Melliar-Smith: Using Interceptors to Enhance CORBA. IEEE Computer 32(7): 62-68 (1999)
[27] Michel Cukier, Jennifer Ren, Chetan Sabnis, David Henke, Jessica Pistole, William H. Sanders, David E. Bakken, Mark E. Berman, David A. Karr, Richard E. Schantz: AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects. Symposium on Reliable Distributed Systems 1998: 245-253.
[28] Aniruddha Gokhale, Balachandran Natarajan, Joseph Cross, and Douglas C. Schmidt, Towards Dependable Real-time CORBA Middleware, Cluster Computing: the Journal on Networks, Software, and Applications Special Issue on Dependable Distributed Systems, edited by Alan George.
[29] The Berkeley/Stanford Recovery-Oriented Computing (ROC) Project. http://roc.cs.berkeley.edu/.

Table 1: Comparing some of the features in WS-Reliability and WS-ReliableMessaging

	WS-Reliability	WS-RM
SOAP related issues	Is a SOAP based protocol, which has an HTTP binding which facilitates acknowledgements and faults to be issued over HTTP responses.	WSRM provides an XML schema for elements needed to support the reliable messaging framework. The specification provides a SOAP binding for the protocol.
Related Specifications	SOAP, WS-Security	WS-Policy, WS-Security
Unique Ids	URI based [RFC 2396], the syntax for the message-ID should be based on what is outlined in RFC2392.	URI based [RFC 2396]. No additional requirement. Messages within a sequence are identified based on message numbers.
Sequence numbering initialization	Starts at 0 for the first message in a group.	Starts at 1 for the first message in a group.
Sequence numbering rollover	Generate a new group identifier and begin new sequence only after receipt of last message in old sequence.	No new sequences can be generated. `MessageRollover` fault is issued.
Presence of numbering information and its relation to delivery	REQUIRED only for guaranteed ordering.	Message number is REQUIRED for every message that needs to be delivered reliably.
Acknowledgements	Can be sent upon receipt of messages, as a callback or in response to a poll. Needed upon receipt of every message.	Acknowledgements can be based on a range of messages, and the timing for issuing this can be advertised in a policy. An endpoint may also choose to send acknowledgements at any time.
Requesting acknowledgements	The `AckRequested` element is REQUIRED in every message for which reliable delivery needs to be ensured.	`AckRequested` is used to request the receiving entity to acknowledge the message received. This is not REQUIRED.
Correlation associated with an Acknowledgement	The identifier associated with the message being acknowledged.	The identifier associated with the sequence of messages and the message number within that sequence.
Timestamps	Are expressed as UTC and conforms to a [XML Schema Part2: Data Types] `dateTime` element.	No explicit reference to UTC. Uses the `dateTime` format.
Retransmissions	Triggered after receipt of a set of acknowledgements. In the event an acknowledgement is not received, the message is retransmitted until a specified number of resend attempts have been made.	Allows the specification of a `RetransmissionInterval` for a sequence (effects every message in the sequence). The interval can also be adjusted based on the exponential backoff algorithm.
Quality of Service	Is initiated by the sender.	WS-Policy assertions are used to meet delivery assurances.
Delivery sequences supported	Exactly once ordered delivery, reliable delivery. Order is always tied to guaranteed delivery and cannot be separately specified.	At most once, at least once and exactly once. Order is not necessarily tied to guaranteed delivery.
Security	Relies on WS-Security and assorted specifications	Relies on WS-Security and assorted specifications

Fault Codes supported by protocol	`InvalidMessageHeader`
	`Invalid MessageIdentifier`
	`InvalidReferenceToMessageId`
	`InvalidTimeStamp`
	`InvalidExpiryTime`
	`InvalidReliableMessage`
	`InvalidAckRequested`
	`InvalidMessageOrder`
		`SequenceTerminated`
		`Unknown Sequence`
		`InvalidAcknowledgement`
		`MessageNumberRollover`
		`LastMessageNumberExceeded`