An Analysis of Reliable Delivery Specifications for Web Services

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

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 (WSR) from OASIS and WS-ReliableMessaging (WSRM) 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 recommendations regarding 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 ease 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, advertisements, discovery 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 (hereafter WSR) [4] from OASIS and WS-ReliableMessaging (hereafter WSRM) [5] from IBM and Microsoft among others. 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 recommendations to address issues identified in the gap analysis. We believe it is quite possible that these specifications may continue to exist alongside each other. To account for such a scenario we also include strategies 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 the area of reliable delivery. In section 3 we include a background 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 issues in these specifications, while providing recommendations to plug these gaps. Finally, we present issues in federating these schemes and outline conclusions.

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 and message queuing systems. We also review fault tolerance approaches in distributed object based systems and 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 the presence of a 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 there is 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 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 background 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 gets positive acknowledgments (ACKs) from all receivers periodically. A *positive acknowledgement* 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. 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 – WSR and WSRM – 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 use positive acknowledgements to ensure reliable delivery. This in turn implies that error detections, initiation of error corrections and subsequent retransmissions of "missed" messages can be performed at the sender side. 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 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* (also referred to as a *sequence*) 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 for both messages (application and control) and group of messages. Individual and group of messages are considered invalid upon the expiry of timers associated with them. Finally, the delivery protocols in the specifications also incorporate the use of 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 security. 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. Difference in approaches

In this section we compare the difference in the approaches and philosophies towards some of the key concepts in these specifications.

5.1. SOAP & HTTP related issues

WSRM specifies an XML based schema for elements that are needed for reliable messaging and includes a SOAP binding for its protocol. WSR, on the other hand, is a SOAP-based protocol for the reliable delivery of messages. WSR also includes a HTTP binding for various messages exchanges that are allowed in the protocol. This includes outlining the HTTP codes, use of the SOAPAction attribute and the strategy to use HTTP Responses for protocol exchanges (including faults). WSRM does not include an HTTP binding. It is expected that WSRM endpoints will use SOAP's HTTP binding.

5.2. Role of intermediaries in the protocol

WSR explicitly forbids the presence of any SOAP intermediaries in the implementation of its specification. This implies that it is not possible to delegate any of the operations related to the protocol, for e.g. the logging process cannot be delegated to any other node. WSRM does not preclude such strategies.

5.3. Message exchange patterns

WSR defines several message exchange patterns (MEPs) related to the protocol. These MEPs leverage the exchange patterns available using SOAP viz. one-way MEP and Request-response MEP. The MEPs in WSR pertain to the strategy to enable a receiver to respond with Acknowledgements or Faults related to received messages. The patterns currently supported in the WSR specification include: *Response*, *Callback* and *Poll*. As we will describe in a later section, a sender can specify the MEP to be used for acknowledgements or faults in every message that it sends. WSRM does not specify any specific MEPs in its specification.

5.4. Grouping and Numbering of messages

In both the specifications every individual message is part of a group of messages. In WSR this group is referred to as a `Group` while in WSRM this is referred to as a `Sequence`. Individual messages also have numbering information associated with them. This numbering information monotonically increases within a group of messages. In WSR this numbering is referred to as a `SequenceNumber` while in WSRM this is referred to as a `MessageNumber`. In both the specifications the identifier for messages is a combination of the group identifier and the numbering information associated with the message. The specifications also include an element which facilitates the tagging of a message as the last message within a group of messages.

In WSR, if a group of messages comprises only one message this numbering information need not be present. In WSRM, on the other hand, every message needs to have this numbering information. This numbering begins at **0** for WSR while it begins at **1** for WSRM. 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 WSRM specification this value needs to be explicitly updated (since a value of zero is an 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 WSR, where it is conceivable that a message can be published without incrementing the value assigned by default. In general it is preferable that the numbering information associated with a message is explicitly increased prior to issuing the message instead of doing so after sending the message across.

5.5. Creation & Termination of Message Groups

WSR includes no specific exchanges to facilitate the creation of a message group. Simply the presence of a new group identifier signals the beginning of a new message group. This scheme can lead to problems if there are collisions in group identifiers at the receiver side. It is entirely possible that two senders may have the same group identifiers for their messages. Depending on the application in question this can lead to unpredictable behavior. This will not be an issue if all endpoints involved in the protocol use identifiers that are based on implementations of the UUID specification, which guarantees unique identifiers up until the year 3040 AD.

WSRM has specific message exchanges that deal with the generation of group identifiers. Furthermore, unlike WSR in WSRM the group identifiers can be generated at the receiver which ensures that there are no namespace collisions at the receiver. If for some reason this causes a collision at the source, the source can issue another exchange to create a new group identifier.

WSR has several rules regarding the termination of a group of messages. These rules also outline when a group is considered complete and when it is ready to be closed. It should be noted that WSR outlines these rules for the sender and receiver separately. These terminations are based on

- Expiration –- The time at which a message group is set to expire
- Idle Timeout – The minimum of time for which there have been no exchanges (messages, control or fault) that were exchanged pertaining to this sequence.
- Completeness –- If all the messages that have been published are both received and acknowledged.
- Sequence exhaustion –- If the message number exceeds the maximum Long value i.e. 18,446,744,073,709,551,615, the sequence is terminated.
- Ordering failure – if there has been a failure in the ability to deliver ordered messages.

WSR however does not have an explicit exchange pertaining to the termination of messages.

WSRM has an explicit exchange between the sender and receiver pertaining to the termination of a sequence. In response to the receipt of all messages (the last message is determined by a special attribute in the message), a receiver sends a control message acknowledging ALL messages in the sequence. Upon receipt of such a message the sender issues a `TerminateSequence` message which indicates the termination of the message group. Terminations also occur due to group expiry and due to message rollovers.

5.6. Acknowledgements

Both specifications facilitate the inclusion of an element that requests acknowledgement for delivered messages. In WSR a sender can also specify the MEP that needs to be used while sending back acknowledgements. The Poll MEP can request acknowledgement for not just one, but

for several groups of messages. Furthermore, in the `PollRequest`, element for every group multiple ranges (corresponding to message numbering such as 0-50, 55-6000) can also be specified. The ranges are optional if the group comprises only one message. Responses to a `PollRequest` can contain information regarding multiple groups. For a given group the range of messages acknowledged corresponds to messages that have the same delivery status. Finally, acknowledgements in WSR also include information regarding messages that triggered faults at the receiver.

In WSRM a receiver sends acknowledgements based on a previously agreed upon `Acknowledgement Interval`. Acknowledgements can include range of messages, however unlike WSR acknowledgements in WSRM pertain only to a specific group of messages. Furthermore, these acknowledgements do not indicate the fault status corresponding to any previously issued messages. If an acknowledgement is explicitly requested by a sender, the receiver is expected to acknowledge all previous unacknowledged messages. The last message in a group, triggers the acknowledgement of the entire set of messages that were received for that group, irrespective of whether they have been previously acknowledged or not. This is a precursor to group termination.

5.7. Error Corrections and Retransmissions

In WSR retransmissions are always initiated by the sender side. This is an artifact of the use of *only* positive acknowledgements in WSR. Since WSR does not support negative acknowledgements, error corrections are always initiated at the sender. Retransmissions can also be proactive where multiple successive attempts at varying intervals (usually increasing) are made to deliver a message. In WSR the retransmissions are triggered faster since an acknowledgement is expected corresponding to the delivery of every message. The sender also attempts retransmissions until a specified number of attempts have been made.

WSRM incorporates support for negative acknowledgements. This means that error detection can be performed at the receiver, and requests for retransmissions can be initiated by the receiver. WSRM also allows the specification of a `RetransmissionInterval` (specified in milliseconds) for a message group. 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. This relates to the time that is allowed to elapse after which the non-receipt of a acknowledgment corresponding to the message triggers retransmissions. The specification also allows this interval to be adjusted based on the exponential backoff algorithm. In WSRM the efficiency of error corrections is determined based on the negative acknowledgements, acknowledgement policy and the time specified in the `RetransmissionInterval.`

5.8. Delivery Modes supported

In WSR the delivery modes supported include: unreliable, at-least-once, ordered-and-exactly-once. Furthermore, ordered delivery is always tied to guaranteed delivery and duplicate-detection. In WSRM the delivery modes supported include: At most once, at least once, ordered and exactly-once. Note that duplicate detection, reliable delivery and ordered delivery can all exist independently within WSRM. Finally, in both the specifications the delivery modes are valid only with a group of messages.

5.9. Policy and protocol constants

WSRM uses WS-Policy to specify protocol constants and other elements that relate to specifying the Quality of Service needed by received sequences. WS-Policy enables an endpoint to describe and advertise its capabilities and/or requirements, and enables communication regarding the characteristics that apply for a given sequence of messages. The WSRM specification also recommends the use of WS-MetadataExchange to exchange these policy elements. In WSR this is done through an abstract concept referred to as an Agreement. The specification does not recommend a specific scheme to achieve this. It is conceivable that one can use the WS-Policy to implement these agreements.

5.10. Fault reporting

WSR outlines two sets of faults viz. message format faults and message processing faults. This covers the entire gamut of errors that could take place while processing exchanges at both the sender and receiver. WSRM also defines a large number of faults covering specific errors that take place during processing. The fault reporting in WSR follows the MEP requested in the original message. Furthermore, during acknowledgements a WSR receiver also indicates the messages that resulted in faults. In WSRM faults are issued as an when they occur; acknowledgements do not include information regarding faults. However, WSRM Fault processing is more sophisticated since one can leverage WS-Addressing's message information headers.

6. Issues and Recommendations

In this section we outline some of the issues that should be addressed in both the WSR and WSRM specification.

6.1. Issues

Both the specifications do not provide for ordered reliable delivery across sequences. The specifications also 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.

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 service **A** 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.

6.2. Recommendations

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.

It should be noted that this does not impose total order on all message issued from a source to a receiver. It simply implies that ordering across multiple sequences is possible, if so desired, by the receiver. Specification of causal ordering relationships between messages across sequences is now easy to specify, verify and satisfy.

To support cases where a rollover might occur at a destination, we also include a *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.

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 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 different 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 previously 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).

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

7. Federation between 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*. Here we briefly review some of the key issues involved. 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, based on the WSR specification.

Exchange of protocol constants related to expiry and the mapping of constants such as exponential backoff intervals by modifying the retransmission interval are two areas that need to be considered. Similarly, group creation and termination strategies need to be mapped effectively so that the behavior is consistent with the expected protocol semantics dictated by each specification. MEPs such as `PollingRequest`s in WSR will trigger multiple acknowledgement requests and responses from the WSRM side. These need to be accumulated by the middleware prior to delivery of the expected `PollResponse` at the WSR side. Finally, management of numbering information on both sides is important. The maximum numbering information in a group would be reduced by 1 to account for the fact that sequence numbers in WSR start at 0 and not 1. All acknowledgement requests and responses between endpoints need to have the numbering information either incremented or decremented depending on the direction of exchanges.

Finally, the faults reported by either side needs to be mapped accordingly. In most cases, there are matching faults that can be found in the two specifications. The middleware needs to keep track of the faults that were issued, this would then used while responding to PollRequests initiated by the WSR endpoint.

7.1. Deployment of the federation module

To facilitate incremental addition of capabilities to service endpoints one can also configure *filters* (examples include filters for encryption, compression, logging etc.) in the processing path between the service endpoints. Since the service endpoints communicate using SOAP messages

these filters operate on SOAP messages. Several of these filters can be cascaded to constitute a *filter pipeline*. In Java these filters are referred to as JAX-RPC *handlers*, in gSOAP they are referred to as *plugin*; while in Microsoft's WSE these are referred to as *filters*. The federation module can be implemented as a filter and configured during the deployment phase of the service in question. Note that this filter strategy while not entail any changes to the service implementations and applications using either specifications. Another deployment strategy is to implement the federation as a proxy, which receives messages and routes mapped messages appropriately.

8. Conclusions

The specifications pertaining to the reliable delivery of messages are one of the most important set of specifications in the Web Services 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 WSR and WSRM have been summarized in the form of table in Table 1 on page 8.

9. 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, et al. 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. 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 Effective Multicast programming 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-ReliableMessaging
Defines	Defines elements and attributes in the header block of a SOAP envelope.	An XML based schema for elements that are needed for reliable messaging.
Related Specifications	SOAP, WS-Security	WS_addressing, WS-Policy, WS-Security
Companies Involved	Sun, Oracle, Fujitsu	IBM, Microsoft, BEA
Submission Status	Under consideration by OASIS to be a standard	Not submitted yet.
HTTP Bindings	Defines a separate HTTP binding of the protocol.	NO. Defines no such binding. However, SOAP's HTTP binding can be leveraged.
Delivery modes supported	Unreliable, at-least-once, ordered-and-exactly-once	At most once, at least once, ordered and exactly-once.
Groups of messages	Identified by `GroupId` information associated with every message in sequence. Individual messages have numbering that increments sequentially.	Grouped together using `Sequence` element. Every Sequence element has a unique identifier, and a message number which increments sequentially.
Dedicated exchanges for creation and termination of message groups	NO	YES
Ordering/Duplication dependence	Order is always tied to Guaranteed delivery and cannot be separately specified.	Order is not necessarily tied to guaranteed delivery
Exchange/Specification of protocol constants	Through an abstract concept referred to as Agreement.	WS-Policy.
Message Numbering in a group of messages.	`SequenceNumber` starts at 0 for the first message in a group.	`MessageNumber` starts at 1 for the first message in a group.
Defines Message-Exchange patterns	Request/Response, One-way and Polling	Not defined
Negative acknowledgements	NO	YES. This enables receiver-initiated error corrections.
Acknowledging a range of messages	YES	YES
Acknowledgements for multiple Groups	YES	NO
Indication of faults in acknowledgements	YES	NO
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.
Time based expiry	Supports timer based expiry of both messages and groups.	Supports timer based expiry of both messages and groups.
Retransmissions	Triggered after receipt of a set of acknowledgements. A specified number of retry attempts are made.	Triggered by either positive or negative acknowledgments. Two other constants `Retransmission Interval` and `exponential backoff` . play a role
Security	Relies on WS-Security and assorted specifications	Relies on WS-Security and assorted specifications
Errors	Are notified through SOAP faults.	Are notified through SOAP faults. Fault processing is more sophisticated since one can leverage WS-Addressing's message information headers.