Will the Real Reliable Messaging Please Stand Up?

A.K.A. WS-Reliability, WS-ReliableMessaging, or WS-ReliableConundrum?

Dave Chappell

Open standards for reliable Web services messaging, such as WS-Reliability, can provide the missing link to bridge the gap between organizations and help make Web services a truly enterprise-capable technology for standards-based systems integration.

Along with security, reliable asynchronous communications has been one of the gaping holes in today's Web services architecture. Lack of reliability, due to the inherent nature of using SOAP over protocols such as HTTP, is one of the biggest obstacles to the adoption of Web services for mission-critical communications between applications and services, such as complex business-to-business transactions or real-time enterprise integration. Standards-based reliable messaging is a cornerstone of the rapidly emerging standards-based integration technology category known as the "Enterprise Service Bus" (ESB).

In fact, the need for open standards for reliable Web services has become so widely recognized that we now have 3 competing sets of SOAP-based reliable messaging out there. All were recently announced, and all are named under the defacto branding of "WS-*". There is the OASIS WS-Reliability spec that a large number of vendors, including Sonic, is a part of. There is a one-vendor set of specs announced by BEA recently, known as WS-Acknowledgement, WS-Callback, and WS-MessageData. Then less than 2 weeks after that announcement, along came another competing specification announced jointly by Microsoft, IBM, BEA, and TIBCO, known as WS-ReliableMessaging and WS-Addressing. So it would seem that BEA is even competing with itself in this area.

On the plus side, it's a pleasure to see that reliable message-based communications—a subject that has long been very near and dear to my heart—has finally become a widely recognized requirement for Web services. So much so that we are witnessing a "land grab" for mindshare and thought leadership in this area. The world's largest vendors are doing battle in the press, making claims about superior technical prowess, and making accusations about being "proprietary". The downside is that the current situation of multiple overlapping specifications is bound to cause further fracturing in the marketplace. The chair of the OASIS Reliable Messaging Technical Committee, Tom Rutt of Fujitsu, has publicly stated an open invitation for all of these to converge under the OASIS TC, and remains optimistic about that happening. Unfortunately, politics may prevail. What the user community, and most of the vendor community, wants is one spec that everyone can point at and feel good about, so we can all move on to larger issues.

The "Superior" spec

After careful examination, I have come to the conclusion that all of the new specs by-and-large cover the same ground. They all do SOAP-based reliable messaging via acknowledgements, and

have varied levels of Quality of Service (QoS) options like at-most-once and at-least-once. All have an ability to specify a URI (URL) as a place to receive asynchronous callbacks, and a message ID based mechanism for correlating asynchronous requests with asynchronous "responses". The smaller differences include things like duplicate-elimination, acknowledgement timeouts, or purported support for WS-Security.

The following is a summary of the three initiatives. They are split up into components of as many as specs in each case:

WS-Reliability -

The WS-Reliability spec itself was announced in January 2003. The announcement of the formation of an OASIS Technical Committee (TC) happened in shortly thereafter in February 2003. Members include CommerceOne, Cyclone Commerce, Fujitsu, Hitachi, Infosys, Iona, Javector Software, NEC, Nokia, Oracle, SAP, SeeBeyond, Sonic, Sun, Sybase, webMethods, WRQ. Just in case its not confusing enough--the name of the draft spec is called "WS-Reliability", and the name of the TC is called "Web Services Reliable Messaging", or WS-RM.

The initial WS-Reliability specification is a draft that was purely intended to be a starting point as input into the formation of the OASIS TC. A number of the things that the other specs address, like WS-Security support, multi-hop routing, and more details on the semantics of timeouts and retries, are all things that the initial authors of the WS-Reliability spec had realized needed to be addressed. However, the consensus was that we should wait for the establishment of the formal TC and work it out there when more companies could get involved. For the most part these things are captured in the "issues" section of the spec. The TC charter also clearly identifies that things like end-to-end reliability and policy frameworks are issues that it wants to address.

WS-ReliableMessaging/WS-Addressing -

A pair of specs that are a joint MS/IBM/BEA/TIBCO effort. Announced on Thursday, 3/13/2003, WS-ReliableMessaging is very similar in form to the draft OASIS WS-Reliability spec. There have been some claims by the authoring companies that this set of specs is superior because they have ties to the WS-Policy framework and WS-Security specifications. WS-Policy and WS-Policy-Assertions are a formal way of asserting capabilities, such as Quality of Service (QoS) and security, and carrying that type of information in SOAP headers. WS-Policy is a good idea, but at the moment the WS-Policy* specs have not been submitted to any standards body. By the time this goes to print that may have changed as well. In any case, lets look at it for what it is. Having support for WS-Policy* really just means that a <AckRequested> tag in WS-ReliableMessaging conforms to the idiom established by the WS-Policy* set of specs. In contrast, the WS-Reliability spec simply saying that <AckRequested> is a header tag. In defense of WS-Reliability, the WS-Policy* support is something that could easily be added to WS-Reliability with as much as an editing pass, if that's what the TC decides to do. At the moment the TC has issues with pulling in specifications that aren't recognized as part of any standards body effort.

WS-ReliableMessaging does have ties to WS-Security, although the way that its done is simply to have a section that calls out some things where WS-Security ought to be applied.
WS-Addressing is a companion spec which defines things like From: and To: in the SOAP header in a way that can retain that type of information across multiple protocol boundaries or intermediary hops.

WS-Acknowledgement, WS-Callback, and WS-MessageData -

This is BEA's proprietary set of reliable messaging specs, which were announced at the opening day of BEA e-World user conference on 3/2/2003, a week and a half prior to the joint MS/IBM/BEA/TIBCO WS-ReliableMessaging/WS-Addressing announcement. This set of specs is a group of 3 that collectively define SOAP-based reliable messaging. Here is a brief description of the role each one plays.

WS-Acknowledgement - Similar in form to OASIS WS-Reliability and WS-ReliableMessaging. SOAP-based reliable messaging with acknowledgements and QoS options like at-most-once and at-least-once.

WS-Callback - A SOAP header mechanism for specifying a URI (URL) for the location of a callback listener.

WS-MessageData - A very simple spec for naming headers that specify message IDs and correlation ID's (RefToMessageID).

In this article, I'm not going to focus too much on the one-vendor set of specs, since I'm willing to bet that even BEA isn't sure what they're going to do with that situation going forward. Although having read through them, the short answer is they are pretty much the same as the other two efforts.

Implied Behavior

Another thing that these specs are they have in common is there is a great deal of implied behavior and "reading between the lines". I'll be the first to admit that about the OASIS WS-Reliability as well. I often get questions for clarity about things that I thought were pretty obvious in the language of the spec ☺. The intent of the WS-Reliability authors since the beginning was to create a draft spec to serve as input into the formation of the OASIS Technical committee. There are lots of unanswered questions, many of which are addressed in the "Issues" section. During the formation of the WS-Reliability spec, we raised lots of issues that in particular had to do with the behavioral semantics of the underlying sending and receiving message handlers, and decided to defer to the full formation of the OASIS TC and let the details be fleshed out with a larger community of vendors and other contributors. And by the way, WS-ReliableMessaging isn't perfect either. Both specs have clarity issues to deal with.

To illustrate this point, let's take as an example the requirement of supporting guaranteed ordering of messages. It's one thing to have header tag definitions for message sequences, but there's much more than meets the eye when it comes to fully spelling out the rules for what the behavior of the

receiving message handler is supposed to be when it encounters a situation where a gap is detected in a group of messages. Both WS-Reliability and WS-ReliableMessaging have provisions for detecting this condition and resending the missing message. WS-ReliableMessaging has much more detail spelled out with regard to this, while WS-Reliability states it in the "issues" list as something that needs to be more fully fleshed out by the OASIS TC. Another example is acknowledgement handling. Both specs have an "AckRequested" header, although in WS-ReliableMessaging its unclear if the <AckRequested> tag is required all the time, or only for the case where an acknowledgement is lost or missing. The spec seems to imply the latter in one of its diagrams, but it never really comes out and explicitly states this. In reality, its really a "Re-Ack" request. I am communicating with the WS-ReliableMessaging spec authors on this, so hopefully this will be cleared up in a future revision of the spec.

Reliable Messaging via Acknowledgements

The common thread across these specs is the notion of verifying the delivery of messages through the use of an acknowledgement message from the receiver. There is an assumption that there is a message handler layer that takes care of this, as shown in figure 1.

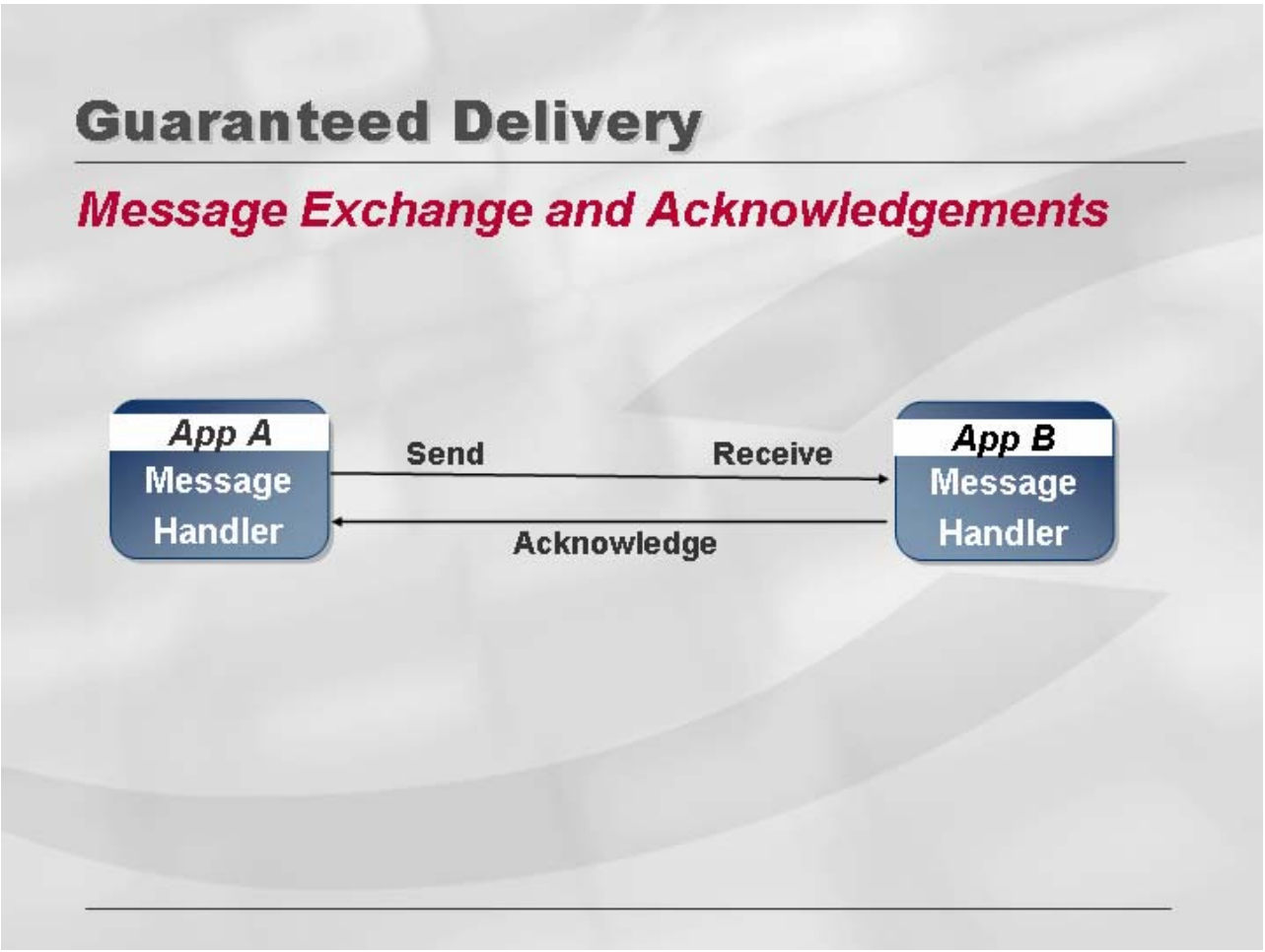

Figure 1: Message handlers achieve reliability via acknowledgements

One of the things that WS-ReliableMessaging has in its favor is the notion of group acknowledgements. A single acknowledgement message may be sent which contains a range of message sequence Ids, including those that are non-contiguous. Furthermore, it appears possible for acknowledgement messages may by attached to other business-level response messages. WS-Reliability simply states in the "issues" section that the behavioral semantics of the receiving message handler need to be further specified with regard to groups of ordered messages.

Another thing WS-ReliableMessaging has in its favor is the notion of a retransmission interval and an acknowledgement timeout. Presumably these two can be used in conjunction with each other to keep the number and frequency of message retries to an optimal minimum, allowing a sliding window of time where multiple acknowledgements can be batched together in a single message in the case where there are no other business level response messages to attach the acknowledgements to. This is another one of those "read between the lines" issues, and one that must be approached with caution. If the retransmission interval is an equal or smaller time interval than the acknowledgement timeout, then too many retries can occur, or retries and acknowledgements can cross over each other simultaneously. Determining the correct ratio between these two time settings is key. I have made a request of the spec authors for more clarity on this, and hopefully it will be addressed in future versions of the spec.

In contrast, WS-Reliability simply states that the retries are configurable, and that a Fault should be raised after a certain configurable number of retries.

Message Persistence

Message Persistence, combined with message acknowledgement, has been a key concept of reliable messaging since Message Oriented Middleware (MOM) has been in existence, and is a critical factor in the new ESB architecture as well. Message persistence allows a sending application to "fire-and-forget" a reliable message, and delegates to the underlying message handler the responsibility of getting the message to its destination (Figure 2).

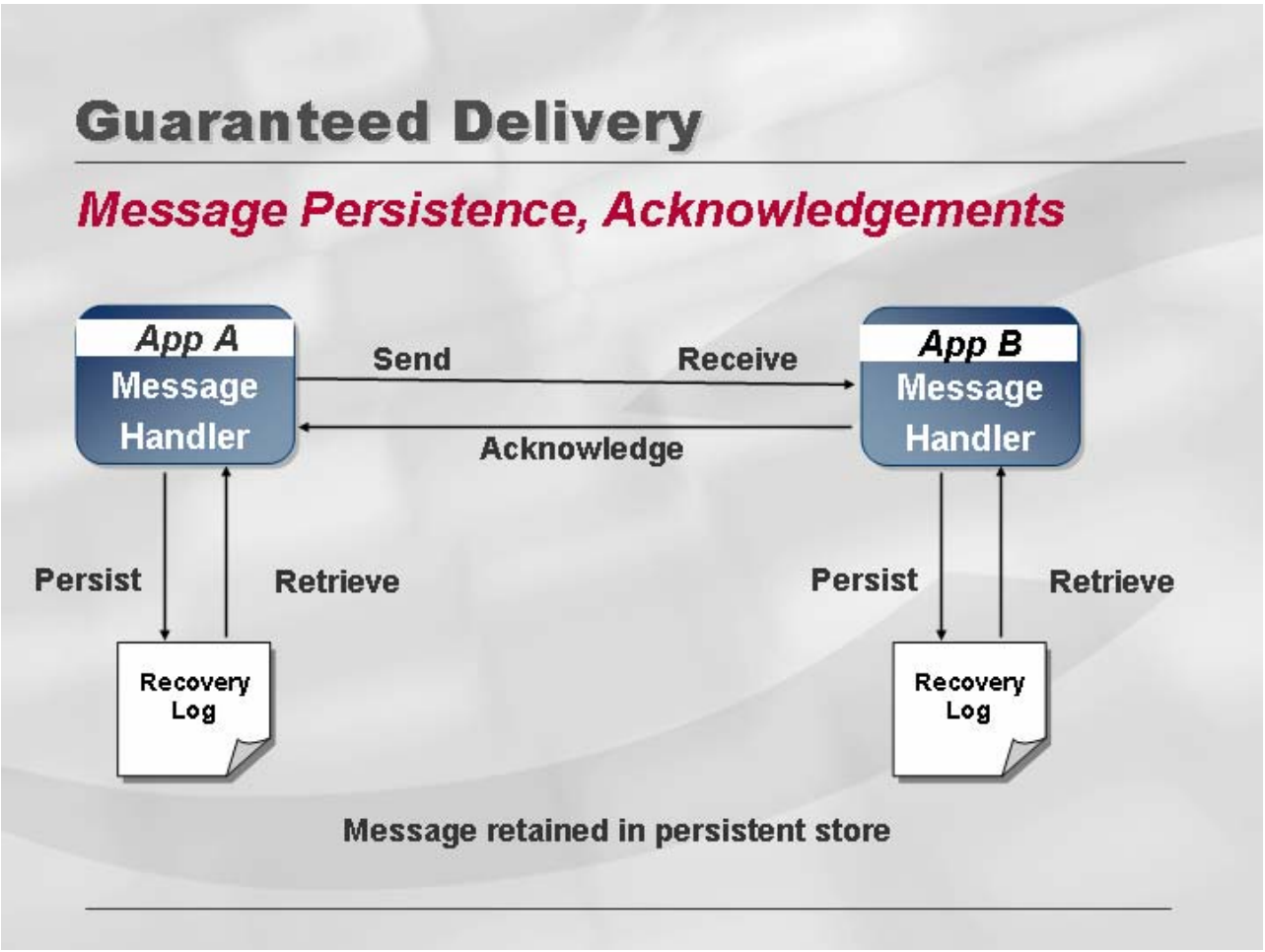

Figure 2: Message Persistence

Message persistence combined with an acknowledgement contract between the sending application, the persistence mechanism, and the receiving node makes recovery possible should the sending application, the message handler, or the receiver fail for any reason during the send operation. Likewise, having a similar contract from the receiver's perspective can make the processing of ordered groups of messages much more capable of recovering from failure. The importance of persistence can be further magnified when you go beyond just two endpoints and consider a larger business process that spans multiple applications and services as shown in figure 3. Without persistence, a failure or temporary unavailability of one of the applications can affect all concerned.

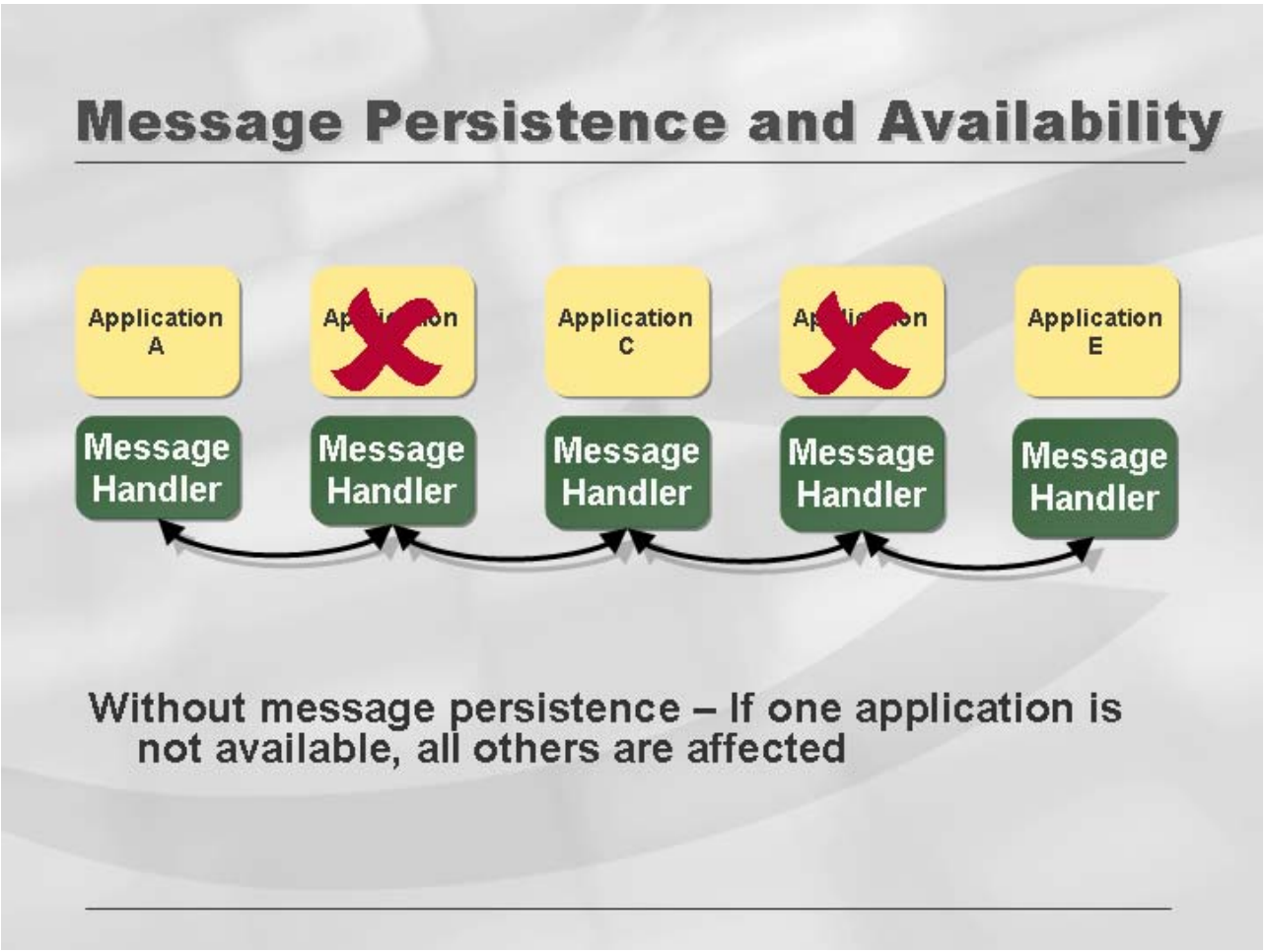

Figure 3: Message persistence can remove the requirement for constant availability.

WS-Reliability states that persistence is a requirement for both the sending and receiving message handler, although it admittedly leaves the details of that as an open issue. WS-ReliableMessaging makes no mention of persistence whatsoever, although there is nothing that precludes it either. It's unclear whether that is an oversight, or the intent. It definitely needs clarity on this. One could imagine a reliable protocol sans-persistence that is purely based on acknowledgements and retries. However the unavailability scenario in figure 3 would result in a fairly complicated exchange of resends and re-acks. Of course, persistence is not without cost. There is always a performance tradeoff to consider when writing things out to disk. Both specs need more clarity in this area.

Comparison Chart

	WS-Reliability	**WS-ReliableMessaging**
Acknowledgement	**Yes**	**Yes**
Message Persistence	**Yes**	**Nothing directly stated**

Message Ordering	**Yes**	**Yes**
At-Most-Once	**Yes**	**Yes**
At-Least-Once	**Yes**	**Yes**
Exactly-Once	**Yes**	**Yes**
TimeToLive	**Yes**	**Yes**
Duplicate Detection	**Yes**	**Indirectly implied by Exactly-Once semantics**
Hop-by-hop Acknowledgement	**No, but recognized by "issues", and by TC Charter**	**Assumed to be covered by WS-Addressing**
Retry	**Yes**	**Yes, with exponential backoff**
Fault Handling	**Yes**	**Yes**
Ack timeout	**No, but recognized by "issues"**	**Yes**
Ack Piggyback	**No, but recognized by "issues"**	**Yes**
Async	**Yes**	**Yes**
Sync	**Yes**	**Yes**
Standards Body	**Yes**	**Not yet**
Message ID	**Yes**	**Yes**
Correlation ID	**Yes**	**Yes**
Callback location	**Yes, via URI**	**Yes, via WS-Addressing**
Security	**No**	**Sort of**
Transaction	**No**	**No**

The Road Ahead

It is unlikely that the competing mega-vendors will concede on this “land-grab” for thought leadership and the race to captivate the developer community with “standards” that boast superior technical prowess. Microsoft and IBM have stated publicly that they fully intend to bring all of the WS-* specifications to a standards body eventually. 2003 will prove to be the “year of the standards shakeout”—or “shootout” depending on how it all unfolds. Lets just hope that you and I don’t get caught in the crossfire.

Even though we had a hand in the crafting of the OASIS WS-Reliability spec, we are going to continue to retain an open and objective viewpoint on all of these specs. Sonic is not interested in getting caught up in the press wars about who has the better spec. Neither are we that concerned over which spec wins. This is not a "In the end….There can be only one" situation. There may actually be two or three! Spec prowess is one thing, implementation is another. Message based protocols require a message based infrastructure. Regardless of what the SOAP header tag looks like, there's still a requirement for a strong messaging infrastructure behind it. Reliable messaging is a tricky business, which takes a great deal of experience to get it right. One of the overarching "read between the lines" aspects of all of this is the requirements of the sending and receiving endpoints, and how they deal with the many intricacies of things like message persistence, failure and recovery scenarios, message redelivery, duplicate detection, in-doubt delivery status, and race conditions with overlapping retries and acknowledgements. The intricacies that can arise out of all these things combined in a day-in, day-out massive deployment environment can be daunting.

Moving On

Reliable messaging is a cornerstone to any enterprise capable integration strategy. Regardless of how this WS-Reliable-Conundrum turns out, the world needs to start focusing on the larger issues. Beyond the base reliable messaging protocol, you still need to think about things like how the messages are orchestrated together, how the XML messages get cached and aggregated, and what the buckets are to place things in when good messages go bad. The rapidly emerging ESB category encompasses these types of issues, and allows for multiple reliable protocols to coexist together. In the end its all about the infrastructure that holds it all together in a platform independent fashion.