RC23720 (W0509-076) September 21, 2005
Computer Science

IBM Research Report

On the Integration of Web Services and Messaging

Ignacio Silva-Lepe, Michael J. Ward, Francisco Curbera
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

On the Integration of Web Services and Messaging

Ignacio Silva-Lepe, Michael J. Ward, Francisco Curbera

IBM T.J. Watson Research Center, 19 Skyline Drive, Hawthorne NY 10532, USA

Abstract. Web Services and Messaging, as application-to-application communication paradigms, have so far been considered separately, with independent programming models and supporting middleware. Different efforts are now introducing messaging notions such as asynchrony, greater consumer cardinality, and looser coupling between web services. This trend will likely result in an extension of the web services programming model. It is not clear, however, that this extension will adhere to a pre-planned approach. A coherent approach requires a thorough integration of the web services and messaging paradigms. This paper proposes one such approach which, in addition to supporting the current style of web services interactions, allows the incorporation of messaging-style interactions under a common programming model. These messaging-style interactions include asynchronous request-response, one-way multi-consumer interactions, and even multiple-choice point-to-point interactions, common in message queuing systems. This paper also elaborates on a model for one-way multi-consumer interactions that integrates the publish/subscribe mode of messaging into the web services programming model. A primary motivation for our approach is to take advantage of key messaging features, while exerting as small an impact as possible on the web services programming model.

1 Introduction

Web Services and Messaging, as application-to-application communication paradigms, have so far been considered separately, with independent programming models and supporting middleware. Messaging, as used here, refers to the approach to application-to-application communication in which the parties, referred to as producer(s) and consumer(s), are not necessarily known to each other and do not have to be available at the same time, and which rely on a third party (not necessarily explicitly instantiated or localized), called a destination, to establish the communication [2, 3]. Distinguishing features of this approach include distribution mode (unicast, anycast, multi-consumer), as well as various qualities of service (reliability, ordering, etc.)

The current state of the practice in web services focuses on client/server, synchronous request-response interactions, primarily over SOAP/HTTP. However, the introduction of specifications such as WS-ReliableMessaging [6], WS-Addressing [4] to a certain extent, WS-Eventing [5], and WS-Notification [7] suggest a trend towards broadening this focus in the direction of asynchrony, greater

consumer cardinality, and looser coupling. For example, WS-ReliableMessaging seems to be motivated by a need for asynchronous one-way point-to-point interactions. WS-Addressing includes an explicit vehicle that can be used to specify asynchronous reponses. WS-Eventing allows the dissemination of events to multiple consumers, which WS-Notification can be seen as extending by introducing brokered (and thus anonymous) delivery. Further, by introducing consumer lists maintained by event sources or notification producers, WS-Eventing and WS-Notification extend the mechanisms to specify the consumer(s) of a web service interaction.

While this trend may ultimately result in an extension of the web services programming model, it is not clear that this extension adheres to an overarching, pre-planned approach. An approach that can fit the current trend, as well as guide the evolution of the web services programming model, may be obtained by an integration of the messaging paradigm with that of web services interactions. This paper proposes one such approach, where web service clients can be seen as specializations of generalized message producers, web service providers can be seen as specializations of generalized message consumers, where messaging destinations are defined as late-binding artifacts, and where the interface between producer and consumer and the makeup of the message to be conveyed are one and the same and given by a WSDL [1] definition. This approach, in addition to supporting the current style of web services interactions, should allow the incorporation of messaging-style interactions under a common programming model. This resulting style of interactions includes asynchronous request-response, one-way multi-consumer interactions, and even the multiple-choice point-to-point interactions that are common in message queuing systems.

This paper also elaborates on a model for one-way multi-consumer interactions amongst web service producers and consumers. This model provides one realization of the proposed approach that integrates the many-to-many messaging mode of interaction (also known as publish/subscribe) into the web services programming model. While incorporating the main features of WS-Eventing and WS-Notification, this multi-consumer messaging model preserves the high-level web services programming model exposed to an application and leaves details such as subscription management to the infrastructure.

2 An Approach to Integration

A primary motivation for the approach proposed in this paper is to take advantage of the multiple cardinality, asynchrony and anonymity features that have made messaging a popular paradigm for a substantial class of applications, while at the same time exerting as small an impact as possible on the programming model that web services applications have grown accustomed to in the last few years.

With this in mind, our approach starts with the web services programming model as it is practiced today. This model consists of web service clients, web service providers and web service definitions expressed in WSDL. A web ser-

vice provider makes itself available by getting deployed at some address. A web service client invokes an operation on a web service provider. This operation is described by a WSDL port type definition. Before a web service client can make an invocation, it gets bound to a reference to the web service provider it intends to invoke.

In order to integrate messaging into this model, we first consider a generalization of this model, and then we cast the elements of the messaging programming model as specializations of this generalized model. This generalization is motivated by the observation that web services applications and messaging applications both engage in application-to-application communication, and thus their basic elements have similar characteristics. A second motivation for this generalization is given by the following observation. An important principle in Service Oriented Architecture (SOA) in general, and in Web Services in particular, is that of late binding of partners. Ideally, the user of a service should not be concerned with what the actual provider is, or even if that provider is determined at run-time, and it should not be concerned with whether the provider is available or not. However, as we have alluded to, in the current state of practice most uses of web services are of the forms (1) targeted, synchronous request-response, or (2) targeted one-way invocations. The term targeted here refers to the coupling of the interaction, where the requester or invoker knows or at some point determines the identity of the provider, whether it be statically or dynamically via a UDDI directory, for instance. Notice that this level of coupling does include a certain degree of late binding; it just does not go as far as the anonymous nature of messaging. This observation leads to the idea that rather than try to introduce messaging destinations as an additional element in the programming model, they could be introduced as an intermediate binding target that delays the binding to the ultimate target.

Thus, our generalization of the web services programming model as we have described it consists of web service message producers, web service message consumers, message definitions expressed in WSDL, and destination bindings. Notice that, since a WSDL operation does not necessarily imply a command semantics[1], a message defintion in this model is actually given by a WSDL operation, and it denotes the schema of the contents agreed to by producers and consumers. A destination binding denotes a reference to an intermediary that enables messages to flow from producers to consumers. Before a web service message producer can send a WSDL defined message, it connects to a destination binding. A web service message consumer makes itself available by connecting to a destination binding at deployment time. Notice that, since web service message consumers receive messages via a destination binding, as long as the denoted intermediary is capable of doing so, more than one consumer may be connected to the same destination binding, just as many producers may connect to and send messages via a given destination binding. Hence, many-to-many messaging interactions are natural in this generalized model.

[1] That is, an operation name does not necessarily denote a task to be performed and can be used instead to annotate the contents of its input message, for instance.

Given this generalization, it is straightforward to see how the web services programming model can be cast as one of its specializations. In particular, a one-way web service client can be seen as a web service message producer for which a one-way invocation consists of sending a WSDL message; a one-way web service provider can be seen as a web service message consumer; the destination binding to which a web service client connects is the actual reference to the web service provider; and a web service provider implicitly connects to itself as its own destination binding by virtue of being deployed. A SOAP/HTTP request-response web service client can be seen as a combination of web service message producer and consumer, where its consumer side and reply-to destination binding are optimized away by the transport; that is, given that an HTTP response is used to carry a SOAP response, there is no need for an explicit consumer or destination binding. Similarly, a SOAP/HTTP request-response web service provider can be seen as a combination of web service message consumer and producer, where its producer side and reply-to destination are also optimized away by the transport.

More notably, given this generalization, the messaging programming model can also be seen as one of its specializations, which we refer to as web services messaging. Here, messaging producers can be seen as web service message producers and messaging consumers as web service message consumers, and the conveyed message is described by a WSDL operation. In addition, a messaging destination, which in this case is an independent third party, is specified by a destination binding. A messaging producer connects to its destination binding before sending messages, and a messaging consumer makes itself available by connecting to its messaging destination at deployment time.

Actual realizations of web services messaging include asynchronous request-response messaging, multi-consumer messaging and multiple-choice point-to-point messaging. Multi-consumer messaging is a form of web services messaging where multiple consumers receive each message sent by a producer, and where the binding destination acts as a mediator to allow their infrastructure to establish either a direct or a brokered connection between them. Multi-consumer messaging is further elaborated on in Section 3.

Asynchronous request-response messaging is a form of web services messaging where a combination web service message producer and consumer sends a request message to an outbound destination and sets up an inbound destination from which its consumer receives the corresponding response. A combination web service consumer and producer connects to the outbound destination to receive a request and binds to the inbound destination to send the response. This form of messaging entails the ability to queue messages at both the outbound and inbound destinations, as well as the ability by the web service producer/consumer to send the request without blocking and to receive the response at a later time by correlating it with the request. It is important to point out that these tasks are not intended to be exposed to the application web service producer/consumer and consumer/producer combinations. Rather, and as we shall also see with multi-consumer messaging, these tasks are intended to be performed by the

infrastructure that underlies the application web services. All that an application web service sees from a programming model point of view is that it can send a request and continue processing until the reponse arrives, and that a request arrives and a response can be sent. The actual details of how this infrastructure is designed are beyond the scope of this paper. Notice that, given the programming model for this form of web services messaging, it may also be possible to derive a targeted asynchronous request-response model, where no third party outbound and inbound destinations are required. Here, the web service producer/consumer outbound destination binding corresponds to the reference to the web service consumer/producer, and the producer/consumer includes in the request a reply-to reference for the consumer/producer to use as inbound destination binding. In fact, this targeted asynchronous request-response model could be seen as a two-way interaction supported by WS-ReliableMessaging, augmented by an infrastructure-supported reply-to consumer that the application producer/consumer can use to obtain its response.

Finally, multiple-choice point-to-point messaging is a form of web services messaging where multiple consumers are eligible to receive any given message sent by a producer to a given destination, but only one consumer actually receives the message. This is a pattern that is common in message queuing systems and that is implied by our approach, provided that the proper infrastructure design is in place for the destination intermediary to select and route any given message to a single consumer. The design of such infrastructure is beyond the scope of this paper.

3 Multi-Consumer Messaging

Web services multi-consumer messaging (MCM) extends the scope of one-way web service interactions to consider more than one web service consumer. To delay the binding of a web service producer to any specific consumer, a message type mediator is introduced. In addition, to increase scalability and to provide multiple possible points of entry for producers and consumers, a broker type of service is introduced. The interface (or message signature) between a web service producer and a consumer is given by the message in an operation of a WSDL port type specification, although it may be possible to specify, on a per consumer basis, the inclusion of a message selector based on the schema of the message.

A message type mediator is a web service that provides an interface that allows the multi-consumer messaging infrastructure for producers and consumers (e.g., producer stub and consumer skeleton) to join the underlying messaging network for the message type.

A producer uses a WSDL document that includes, in its binding and service definition, the message type mediator for the message type it is interested in working with. This WSDL document is referred to as the common WSDL document for the message type. A consumer uses an extension of the common WSDL document that also includes a reference to itself and an optional message selec-

tor. This WSDL document extension can be seen as the consumer's subscription specification.

This way, a producer's multi-consumer messaging infrastructure is able to contact the message type mediator and determine a point of entry into the messaging network. Similarly, a consumer's messaging infrastructure, as part of the corresponding deployment process, contacts the message type mediator to register interest in messages of the given type, and to determine a point of entry into the messaging network.

A broker is an intermediary composite web service that relays messages from producers to consumers and that is used to form the multi-consumer network for a message type.

Given that, in keeping with the web services programming model, the subscription model for a consumer is based on a WSDL message, which is defined by an XML-schema, the subscription model becomes content-based. Notice however that this does not preclude the use of a topic-based subscription model. First, a WSDL message can readily model simple topic categorization and subscription. More advanced hierarchical topic categorization and topic subscription expressions entail the definition of a particular message structure and a subsequent subscription expression dialect.

The following sections elaborate on the multi-consumer messaging infrastructure model. More advanced features, including support for hierarchical topic categorization and subscription expressions, selective subscription propagation, and reliable multi-consumer messaging are beyond the scope of this paper. However it is worth pointing out that the model presented here is intended to provide a foundation for these advanced features. For instance, an MCM network and its message dissemination invariant, as presented in the following section, provide the foundation for tree-shaped message-tracking sequences that are at the core of reliable multi-consumer messaging.

3.1 Multi-Consumer Messaging Infrastructure Model

The multi-consumer messaging infrastructure consists of a number of web services that underlie client application producers and application consumer web services, and that enable the delivery of web service one-way messages from any application producer to any number of application consumers. These infrastructure web services include: multi-consumer messaging producer (or *MCM producer* for short), *MCM consumer*, *MCM broker*, and MCM message type mediator (or *MTM* for short).

An MCM producer is a web service that is given a message to send to one or more MCM consumers. An MCM producer can be given a message by a client application producer or by an MCM consumer. In the former case we refer to the MCM producer as an *lprod*, in the latter as an *inprod*. An MCM producer maintains a list of MCM consumers to which it sends the messages it is given. When an MCM producer p has a reference to an MCM consumer c in its list of consumers, p is said to be connected to c.

An MCM consumer is a web service that receives messages from one or more MCM producers to deliver to either a number of application consumer web services or to another MCM producer. In the former case we refer to the MCM consumer as an *lcons*, in the latter as an *incons*. Messages delivered to an application consumer will be filtered by its MCM consumer according to the message selector specified by the application consumer, if any. An MCM consumer c becomes connected to an MCM producer p by sending a request to p to add c to p's list of consumers.

An MCM broker is a composition of an incons and an inprod, where any data message received by the broker's incons is relayed to the broker's inprod to be sent. Two brokers b_1 and b_2 are said to have a bi-directional connection if b_1's inprod is connected to b_2's incons and b_2's inprod is connected to b_1's incons. Two brokers b_1 and b_2 establish a bi-directional connection with each other when their respective inprods become connected to their counterpart inconses. An lprod lp is connected to a broker b if lp is connected to b's incons. An lcons lc has a connection from a broker b if b's inprod is connected to lc.

An MCM network is a triple $< lp, lc, b >$, where lp is a non-empty collection of lprods, lc is a non-empty collection of lconses, and b is a possibly empty collection of brokers. An MCM network $< lp, lc, b >$ aims at maintaining the following message dissemination invariant: in the absence of failure, a message produced by any lprod in lp will be received by every lcons in lc. To maintain its invariant, an MCM network satisfies the following rules:

1. If it has an empty collection of brokers, then each lprod in lp must be connected to every lcons in lc
2. Otherwise:
   (a) When a broker b_1 joins the network, it must establish a bi-directional connection with a single other broker b_2 in the network
   (b) Each lprod in lp must be connected to exactly one broker in the network
   (c) Each lcons in lc must have a connection from exactly one broker in the network

The diagrams in Fig. 1 illustrate two MCM producers underlying corresponding application producers (i.e., two lprods) and two MCM consumers underlying corresponding application consumers (i.e., two lconses). Fig. 1(a) illustrates the absence of a broker and thus the need for full connectivity amongst lprods and lconses. Fig. 1(b) shows the use of a broker.

As an MCM network evolves, any given broker b_x will typically have bi-directional connections to n other brokers. That is, b_x's incons and inprod will be connected to each of b_i's inprod and incons, respectively, where b_i is one of the n brokers b_x is connected to. For example, in the MCM network of Fig. 2, broker 4 has bi-directional connections with three other brokers.

To ensure that message flow in the network is only 'downstream', when a message arrives at b_x's incons from b_i's inprod, and after being relayed to b_x's inprod, b_x's inprod will make sure that it is sent to b_j's incons, where b_j is each one of the $n - 1$ brokers b_x is connected to, besides b_i. To achieve this, the

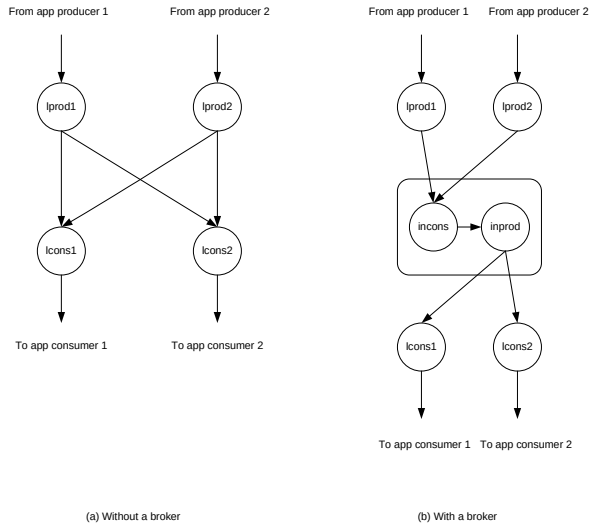

Fig. 1. Broker usage

message must contain a tag that identifies the broker that sends it, b_i in this case. For example, a data message that arrives at broker 4's incons from broker 2's inprod will only be sent to broker 1 and broker 3's inconses by broker 4's inprod, in addition to lcons7, of course.

An MTM is a web service that maintains information about the current state of an MCM network for a given message type; if the network has a non-empty collection of brokers this information includes, for each broker in the network, the endpoint references for the broker's incons and inprod; otherwise, this information includes the endpoint reference of every lprod and every lcons in the network.

An MTM provides an interface for an lprod to join the network and to obtain a list of possible MCM consumers to connect to, annotated with an indicator of whether these are inconses or lconses.

An MTM also provides an interface for an lcons to join the network and to obtain a list of possible MCM producers from which to request a connection, annotated with an indicator of whether these are inprods or lprods.

Finally, an MTM provides an interface for a broker to join the network and to be removed from the network.

3.2 Setting up and Maintaining the Network

At deployment time of an application consumer web service, the corresponding lcons sends a request to the MTM to join the network, including the lcons' endpoint reference in the request. The MTM responds with a list of MCM producers for the lcons to connect to. If the list is annotated as a list of inprods, then the

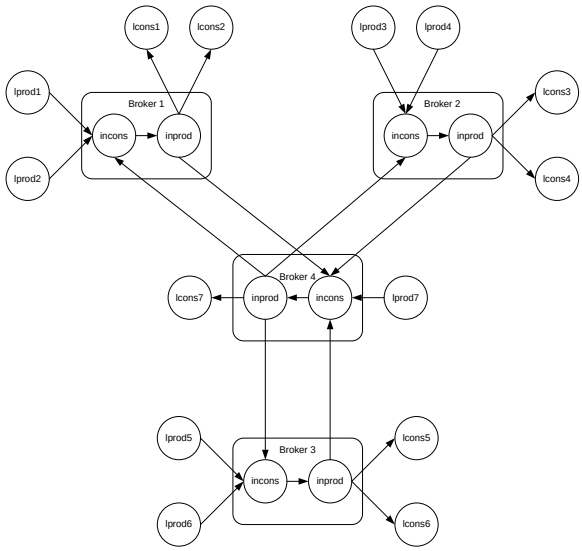

Fig. 2. MCM Network

lcons picks one and sends a request to connect; this request includes the lcons' endpoint reference. Otherwise, if the list is annotated as a list of lprods, the MCM consumer must send a request to connect to each lprod in the list.

Before an application producer client can send a message for the first time, the corresponding lprod must send a request to the MTM to join the network, including the lprod's endpoint reference in the request. The MTM responds with a list of MCM consumers to connect to. If the list is annotated as a list of inconses, then the lprod picks one and adds it to its list of consumers as its single consumer. Otherwise, if the list is annotated as a list of lconses, the lprod must add each lcons in the list to its list of consumers. Note that if the list is empty, the lprod may try to block the application producer client until a consumer is available as, in general, its messages may be lost otherwise. At any time after an MCM producer first connects as an lprod or it does not find any MCM consumer to connect to, it may receive a request to connect from an MCM consumer, which includes the MCM consumer's endpoint reference. Upon receiving this request, the MCM producer adds the MCM consumer to its list of consumers.

At any point before or after any lprods or lconses are added to the MTM's list, the first broker may ask the MTM to be added to the network. If the MTM has any lprods or lconses in its lists, it returns them to the broker and expects an acknowledgement from the broker that the lists can be removed. Upon receipt of a non-empty list of lprods, the connecting broker sends a request to each lprod

in the list to remove its connection to any lcons and to add a connection to the broker. Upon receipt of a non-empty list of lconses, the MCM broker connects to each lcons. The MCM broker then sends an acknowledgement to the MTM. The MTM is then able to remove its lists of lprods and lconses. From the time when the first broker asks to be added to the network until the time it sends acknowledgement to the MTM, the MTM may refuse or delay for a configurable amount of time any other request from a broker, an lprod or an lcons, to ensure that its information about the state of the network remains consistent. If the new broker does not send its acknowledgement within the configured time, the MTM may assume the operation failed and the new broker must try again.

Any time thereafter, a new broker may ask the MTM to be added to the network, to which the MTM responds with a list of MCM brokers for the new broker to choose from. The new broker then establishes a bi-directional connection with the chosen broker.

At any time after a broker has been added to the network, it may receive a request to connect to an lcons or to establish a bi-directional connection with some other broker.

As the network of brokers for a given message type evolves, brokers may be removed. When this happens, the bi-directional connections that a removed broker b_r maintains with other brokers must also be removed. In particular, each such broker must remove b_r from its inprod's list of consumers. Furthermore, the n bi-directional connections being removed must be replaced by $n-1$ bi-directional connections amongst the n corresponding brokers involved in such a way as to preserve the structure that maintains the network's invariant and downstream flow of messages. Notice that is only necessary when $n > 1$. This can be accomplished, for instance, by the following procedure. Let B be the set of brokers with which a removed broker b_r maintained bi-directional connections. While the current B is not a singleton: pick a b_j from B, subtract it from B, and send a notification to b_j to establish a bi-directional connection with some (and only one) member of the resulting B. This procedure may be implemented by the MTM when b_r asks to be removed from the network, where the set B is b_r's list of consumers.

When the last broker asks the MTM to be removed from the network, it includes its lists of lprods and lconses in the request so that the MTM can reconstitute its own lists of lprods and lconses. The removed broker also notifies each lprod to replace its connection to the broker with a connection to each lcons in the broker's list.

4 Related Work

WS-Eventing [5] provides a light-weight facility for disseminating events from an event source to multiple subscribers and to manage the corresponding subscriptions. Since subscribers interact directly with an event source, there is no real decoupling between the two. In addition, an event source can be seen as its own destination with an implicit message type. Presumably, if two web service pro-

ducers wanted to produce events of the same type, they would be independent event sources of their own, with the implication that subscribers would have to subscribe to every event source that ever produced events of a given type. We view WS-Eventing as a foundational architecture on top of which higher-level models, such as multi-consumer messaging, can be built.

WS-Notification [7], in addition to providing a basic WS-Eventing type of dissemination, promotes a brokered many-to-many style of notification. Hence, decoupling between producers and consumers is a main feature. However, WS-Notification is not explicit about how broker networks are to be organized. Without this, building support for end-to-end reliability protocols for large deployments of producers and consumers becomes problematic. In addition, WS-Notification uses topics as its primary means to organize and categorize messages, with message types as optional constraints on topics. We believe that this unnecessarily modifies the programming model, given that the content-based form of message organization, afforded by a straight use of WSDL message types, can be thought of as subsuming topic-based message organization.

Container-Managed Messaging (CMM) [8] is aimed at providing a uniform view of either object-oriented components or message-oriented applications to object-oriented components. To this end, an architecture and programming model are defined that approach messaging as a container-managed service. This is analogous to our approach to integrate web services and messaging from a web services point of view. The CMM architecture includes artifacts such as a message proxy, a result proxy, a message listener, and a callback proxy. Artifacts similar to these would play an important role in the design of an asynchronous request response messaging infrastructure, as we have outlined it in Section 2. CMM focuses more on point-to-point messaging interactions and less on addressing multiple consumers. In addition the CMM architecture is defined solely in object-oriented component terms and not as a generalization that can also cover message-oriented applications.

5 Conclusion

We have presented an approach to integrating web services and messaging, based on a generalization of the web services programming model, of which web services messaging is defined as another specialization. As we have seen, web services messaging can then be realized as modes of interaction that include asynchronous request-response messaging, multi-consumer messaging and multiple-choice point-to-point messaging.

We believe that the integration approach we have presented has the potential of bringing into the mainstream of service-oriented computing the features of message-orientation that have made it a useful, albeit independent, paradigm. Moreover, by integrating messaging with a standards-based platform such as web services, our approach has the potential of leading the way to a standardization of the messaging paradigm itself, independently of any programming language, virtual machine, etc.

References

[1] Web Services Description Language (WSDL 1.1). W3C Note, March 2001.

[2] Java Message Service. `http://java.sun.com/products/jms/docs.html`, April 2002. Version 1.1.

[3] Chuck Cavaness and Brian Keeton. The Components of Java Message Service. `http://www.quepublishing.com/articles/article.asp?p=26270&rl=1`, April 2002. Section 1, Introduction to Messaging.

[4] Don Box, et al. Web Services Addressing (WS-Addressing), August 2004.

[5] Don Box, et al. Web Services Eventing (WS-Eventing), August 2004.

[6] R. Bilorusets, et al. Web Services Reliable Messaging (WS-ReliableMessaging), February 2005.

[7] S. Graham, et al. Publish-Subscribe Notification for Web Services, March 2004. Version 1.0.

[8] Ignacio Silva-Lepe, Christopher Codella, Peter Niblett, and Donald Ferguson. Container-Managed Messaging: An Architecture for Integrating Java Components and Message-Oriented Applications. In *Proceedings of the 37th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-Pacific 2000)*, Sydney, Australia, November 2000.