A Message Oriented Middleware Solution Enabling Non-Repudiation Evidence Generation for Reliable Web Services

Simon Parkin[1], David Ingham[2] and Graham Morgan[1]

1 School of Computing Science, University of Newcastle, NE1 7RU, UK
2 Arjuna Technologies Ltd., Newcastle upon Tyne, NE1 7RU, UK
S.E.Parkin, Graham.Morgan@newcastle.ac.uk dave.ingham@arjuna.com

Abstract: The paper describes an approach to providing reliable message passing together with mechanisms for enforcing non-repudiation for use by Web Services. In particular, we are concerned with message passing that occurs across organizational boundaries and evaluating the suitability of the Java Messaging Service in this approach.

Keywords: Web Services, Middleware, MOM, Non-repudiation

1. Introduction

Business communities have traditionally participated in inter-organizational communications via a number of well known techniques such as face-to-face meetings or paper mail. Two important properties associated with inter-organization communications that contribute to successful commerce are reliable information delivery and the trust in the authentication of the originator of the information. Such reliability stems from the ability of a communications medium to provide a level of guarantee for information delivery that is agreed upon by all participants and satisfies the business function as dictated in some contractual agreement. A signature that all parties agree upon as proof of originator is used to provide trust in the origins of information. Inter-organizational disputes are resolved through some legal action directed by appropriate laws. For example, in law the trust mechanism used to overcome a claim of non-repudiation relating to a communication is the witnessing of signature(s).

Electronic commerce makes possible the implementation of existing business practices via enabling digital technologies. Such technologies ease interaction between organizations and the individual by overcoming traditional problems (e.g., paper based, voice) associated to the geographic distribution of participants involved in a business process. The proliferation of the Internet has contributed to the ability of an enterprise to provide their services to a much larger audience than ever envisaged before the existence of widely available public access networks. Furthermore, the properties of electronic communication (e.g., speed, automation) have brought about business processes that would not be possible using non-digital technologies.

To enable the deployment of applications that span organizational boundaries there is a need to enable interactions between organizations in a manner that does not rely on the specific implementation of an organization's technologies yet can promote interoperability in a heterogeneous environment. One possibility for developing such applications is the Object Management Group's CORBA [10] and related specifications. CORBA is a mature specification that provides interoperability for distributed applications built in a heterogeneous environment and is based on the object-oriented paradigm of program development. However, a service based approach using text based messaging as opposed to CORBA's object-oriented approach with binary messaging is considered more suited to inter-organizational application development [11].

Web Services are promoted as providing a suitable paradigm for application integration across organizational boundaries. Services may be implemented and deployed using platform specific mechanisms with interoperability achieved via Web Service standards and communications over standard protocols. The Protocol specified by Web Services is SOAP [7] (providing RPC) with organizations describing their services, and so making them available to clients, via WSDL [12]. WSDL and SOAP are specified using XML [13]. XML allows a developer to represent different elements of data in a text file that may be read and processed by applications (providing appropriate message descriptions for loosely coupled systems).

We propose the use of *message oriented middleware* (MOM) in a solution to satisfying reliable communications while tackling the problem of non-repudiation for Web Services using SOAP and WSDL. We exploit the message passing properties associated with MOM to prevent partial system failure from inhibiting the delivery of messages and prevent limited transient unavailability of clients and servers from resulting in non-completion of a SOAP RPC. Combining persistent messaging with transactional and security mechanisms aids in non-repudiation. Furthermore, our approach maintains message logs to aid in any inter-organizational disputes relating to non-repudiation that may occur. We have implemented our system using only standard technologies, with clients and servers requiring no amendment to use our system. Our system appears transparent to clients and servers.

The main contribution of our paper is to provide the community with an engineered solution that exhibits the benefits of using MOM for non-repudiation and reliability in the context of Web Services. Our purpose was not simply to implement Web Service standards associated to non-repudiation and reliability (on which there are many works).

In the next section we describe our assumptions related to the technologies we use. Section 3 describes our implementation. Section 4 describes related work with section 5 presenting our conclusions and future work.

2. Background

This section gives a short introduction to SOAP and MOM and explains assumptions we make regarding server/client interaction.

2.1 Clients and Servers

We assume clients enact an RPC on a server using SOAP [7] over HTTP across public access networks (i.e., the Internet). This assumption is based on the fact that SOAP over HTTP is the common configuration for accessing Web Services over the Internet [9]. This is due to the expectation that the use of HTTP is widespread and HTTP is conceptually similar to SOAP as they both describe a request/response style protocol (easing the coupling of these protocols). However, the approach of using SOAP over HTTP is not without problems: the best-effort expectations of HTTP to transmit SOAP messages are not appropriate for some applications which require more robust delivery requirements. For example, inter-organizational interactions via SOAP RPC may require non-repudiation properties that provide a basis for determining the validity of messages (as is the subject of this paper).

The use of SOAP is not restricted to client/server interaction that may necessarily result in request/reply style messaging. SOAP messages may be used in a document-literal style that does not depend on a client invoking a particular method on a server and is therefore message based as opposed to RPC based. Furthermore, a SOAP RPC may not necessarily require a server to generate a reply for every request. In this paper we are primarily concerned with SOAP RPC in which every client request results in a server generated reply, even if this reply is simply an acknowledgement of delivery by the server. This decision has been taken as it is assumed clients require an acknowledgement to enable application level decisions to be made on the successfulness of their request. When an RPC crosses organizational boundaries then only via server acknowledgment may a client be able to state a case that it had understood the request to be delivered if a dispute relating to the delivery status of a message arose between client and server.

We assume servers describe their services via WSDL. WSDL provides a means by which servers may describe their services in a manner that allows clients to contact and use such services. Such a description includes the name of the service, the location of the service (typically a URL), methods available for invocation and the input/output parameter types defined for each method.

2.2 Message Passing

As previously described, SOAP RPC over HTTP is the mechanism we assume clients and servers use to interact. However, the best effort reliability of HTTP coupled with lack of non-repudiation techniques requires a different approach to message passing across organizational boundaries. Therefore, we employ message oriented middleware (MOM) as the basis of our approach for inter-organizational message exchange.

MOM allows two or more applications to exchange messages. The CORBA Notification Service [6] and JMS [1] are examples of specifications that describe typical MOM type services. Unlike RPC, there is no requirement for participants in a MOM message exchange to be contactable at the time of communications. In this sense, senders and receivers of messages are decoupled with receivers consuming messages as and when they are able to. This property may be exploited to provide a means of masking client/server unavailability during the enacting of an RPC. For example, a server may be unavailable to service an RPC (e.g., due to high processing loads, administrative downtime). If an RPC is issued by a client during this period a client may get an exception raised that the server may not be able to process the request or the client may timeout the server if the server is unreasonably slow. Either of these scenarios will result in a client managing its own message resends. Consider this example further. Assume a client timeouts a server and reissues a request. Unfortunately, the server actually processed the original request but was simply too slow in returning a response. This results in duplicate request processing, an undesirable problem in distributed applications, but is a considered a more serious problem for inter-organizational communications where such processing may carry a financial penalty for the client. Overcoming this problem requires agreement between clients and servers on unique identification of requests to allow servers to identify repeat requests. However, in relation to non-repudiation this scheme is not easy to implement across organizational boundaries due to the level of trust and the limited degree of information sharing organizations will tolerate.

MOM may employ additional mechanisms to provide reliability guarantees for message exchange. Atomic transactions coupled with persistent messaging provide fault-tolerance in that the failure of the MOM system or any of the participants in message exchange will not necessarily result in the loss of messages. Atomic transactions are used to ensure the underlying persistent store remains consistent and as long as such a store remains correct and reachable then messages will not be lost. Atomic transactions have an all or nothing property in that an attempted amendment to data is either successfully carried out or not carried out at all. Persistence of messages coupled with atomic transactions is desirable in non-repudiation techniques as failure should not render the system incapable of satisfying the requirements of non-repudiation.

3. Implementation

A Java implementation of our system is achieved via Reliable Routing Nodes (RRNs) and the Java Messaging Service (JMS) [1]. The messaging transport used by JMS is HTTP. An RRN receives client requests and server replies and is responsible for attempting to deliver requests/replies to the appropriate servers/clients. Client requests are uniquely identified within the system to enable the tracking of requests and their associated replies. The JMS provides reliable persistent message storage and forwarding for use by an RRN. Client and server interaction is assumed to be modeled in the Web Services domain with messages described via SOAP and services described via WSDL. An RRN is responsible for maintaining a non-repudiation log for recording requests and their associated responses. This log is persistent in nature and is held in a MySQL database.

Our system may be structured as a single RRN or a network of RRNs. In the single RRN approach all clients and servers are serviced by a centralized RRN that is responsible for handling all messages and associated non-repudiation logs. This approach is suited to systems that may exist within a single organizational domain where administration of the RRN system is not shared. When message transmission spans organizational boundaries an approach that uses a network of RRNs is advocated (figure 1). In this approach an RRN may be placed within each organization with inter-organizational communications mirrored by inter-RRN communications. Additional security measures are taken to attempt to ensure messages are genuine and may be trusted. Administration of RRNs is assumed to be shared amongst organizations (responsible for RRNs within their own domains).

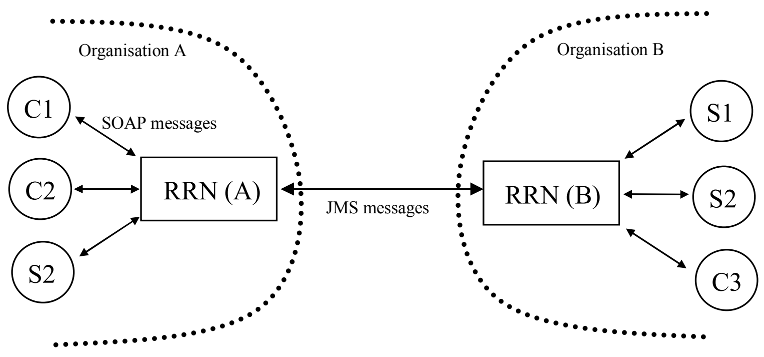

Figure 1 – Network of RRNs facilitating inter-organizational interaction.

A non-repudiation log is amended whenever a message is received or sent by an RRN. This log forms the non-repudiation evidence that may be used in inter-organization disputes regarding requests and replies. The use of reliable persistent messaging between organizations together with security measures provides the basis for enabling our approach to non-repudiation. We now describe each component in more detail. For ease of explanation, we shall only consider a single RRN approach in our descriptions unless otherwise stated.

3.1 Providing System Transparency for Clients

The *client handler* is co-located with a client and intercepts client requests before they reach the underlying transport. This requires no changes to the client implementation and the interception of messages is transparent to client operations via the use of handlers as defined in the Axis toolkit [3]. Therefore, we assume the use of the Axis toolkit in client side application development and deployment.

The Axis toolkit eases the development of Web Service based applications by providing a framework for constructing distributed applications that use SOAP for their message exchange (Axis toolkit is commonly described as a SOAP engine). The Axis toolkit includes support for describing Web Services (Web Services Definition Language (WSDL)) and allows a Web Service Deployment Descriptor (WSDD) to be defined that describes the deployment scenario of one or more Web Services. For example, a WSDD may describe the backend components that are used to implement a Web Service. A WSDD may also describe a chain of handlers which SOAP messages pass through during run-time. The ability of a handler to alter messages is exploited by our system to provide RRN transparency to clients.

The client handler intercepts client requests and performs a series of alterations on the message before allowing the message to continue in transit. A new SOAP entry header is created that records the original target endpoint of the request (the Web Service provided by a server). The original target endpoint of the request is replaced by the endpoint that identifies an RRN. This substitution enables the redirection of the request towards the RRN responsible for handling this client's requests. The type of response expected by a client is checked via the identification of return parameters in a message. From such parameters it is possible to determine if a client knows in advance the expected response. This information is inserted into a new header entry and is later used to determine the appropriate tracking of the message.

3.2 Managing Requests and Replies

The *routing provider* (RProvider) is a Web Service that accepts the re-directed requests issued by the client handler. Requests are formatted to an appropriate message structure for handling by the JMS. Client requests are placed in the *request queue* ready to be consumed and processed by the routing server (RServer). In addition to accepting requests directly from the client handler the RProvider is responsible for returning replies to clients. Replies are gained from the *response queue* (JMS). Therefore, the *routing listener* (RListener) must derive the appropriately formatted SOAP message from the messages consumed from the response queue before returning a reply to a client. Figure 2 shows the flow of messages throughout the components of an RRN.

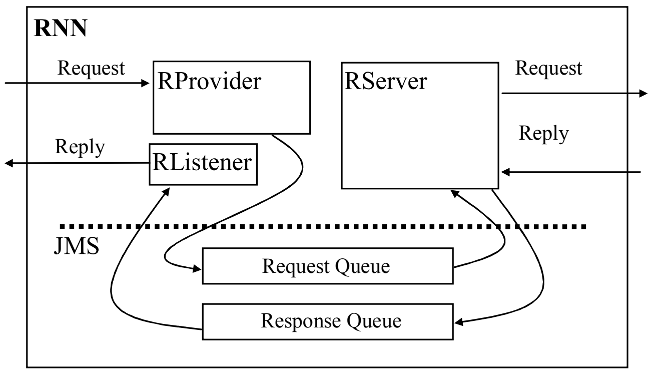

Figure 2 – Components of an RRN.

The RServer consumes messages from the request queue and examines the content of each message to determine the appropriate handling of a message. There are two possible actions the RServer may take based on message contents: (i) attempt to issue request to Web Service endpoint as described in a header entry of the message or; (ii) attempt to forward message to another request queue located in another RRN. In (i) the appropriate SOAP message is created from the contents of the JMS message and issued to a Web Service. Replies generated from a request are then formatted to an appropriate message structure for handling by JMS and placed in the response queue. In (ii) the target endpoint described in a message is looked up in a locally held routing table that identifies the RRN the message should be forwarded to. The routing table is XML based and is held locally on the same machine as an RRN. The successful identification of a target RRN results in the RServer (of the originating RRN) attempting to place the message in the target RRN's request queue. The originator node ID (unique across RRNs) is attached to the JMS message as a message property to enable the identification of the originator RRN by the target RRN (required to ensure a reply may be returned to the originator RRN). Ensuring replies are returned to originating RRNs is the responsibility of the target RRN's RListener. The RListener consumes messages from the response queue that have originator node ID fields set and places such messages on the appropriate originator RRN's response queue (as dictated by the node ID field of the message).

3.3 Undeliverable Messages

Messages that the RServer is unable to deliver to a Web Service (target endpoint) or another RRN's request queue are placed on a *retry queue* (JMS). In the case of an RServer attempting to deliver a message to a Web Service endpoint, messages are identified as undeliverable if exceptions are raised indicating the Web Service is unreachable (either network problems or unavailability of service) or the request timed out. The aborting of the transaction (see 3.4 for more details) within which an RServer was attempting to move a message between request queues indicates an undeliverable message. Periodically messages are moved from the retry queue to the request queue to allow the RServer to attempt message delivery again. The number of retries associated with messages and the frequency with which messages are transferred from the retry queue to the request queue may be set by an administrator of the system. Messages are permanently moved to the failed message queue after the RServer's repeated attempts to deliver the message ended in failure (number of attempts indicated by administrator). When messages are placed on the failed queue information related to why the message failed is appended to the message (e.g., transport exception). The use of retry queues and failed queues by the RServer is mirrored by the response listener in the process of propagating replies back to an originating RRN.

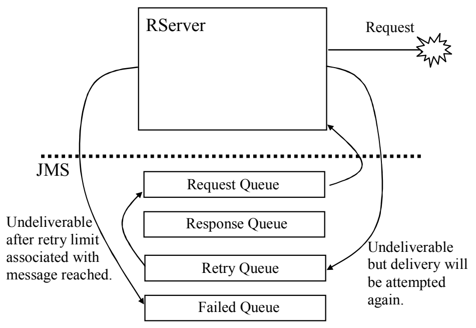
RServer
Request
JMS
Request Queue
Response Queue
Retry Queue
Failed Queue
Undeliverable after retry limit associated with message reached.
Undeliverable but delivery will be attempted again.

Figure 3 – Handling undeliverable messages.

Clients may timeout a request and may not be prepared for a reply when one is available. Furthermore, a client may reissue a request causing duplicate requests to be present in the system. In an attempt to prevent such a scenario the local RRN associates a timeout for each request received. If this timeout expires before a reply is received (consumed by RProvider from *response queue*) a reply is constructed that is in the form of a custom SOAP fault that contains the unique identifier of the related request. This reply is returned to the client. By using this unique identifier in subsequent retries of a request it is possible for clients to retrieve a reply from a request that was previously timed out. This approach does not accommodate client timeouts that expire before an RRN can raise a SOAP fault. However, with clients and an RRN within the same organizational domains we assume it should be possible to tailor the timeout in such a way that clients do not timeout their requests before a SOAP fault may be raised.

3.4 Reliability and Security

Reliable messaging is possible as the JMS specification identifies the ability to ensure guaranteed message delivery even if partial system failure occurs. As described in 3.2, persistent store and delayed message forwarding allow the delivery of messages to endpoints that may suffer transient unavailability (i.e., not able to consume messages as and when messages become deliverable). Furthermore, the persistent nature of the queues ensures that failure of the JMS messaging middleware itself will not lead to the loss of messages (assuming persistent store remains correct and reachable). Our implementation uses the Arjuna Message Service (ArjunaMS) [2], an implementation of the JMS 1.1 specification [1].

Atomic transactions are used whenever message queues are accessed by an RRN. This guarantees that messages are not lost due to RRN failure. If transactions are not available, messages may be lost if an RRN fails after it has consumed a message from one queue before it has placed the same message in another queue.

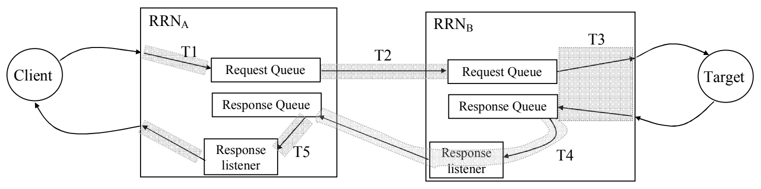

Figure 4 – Transactions satisfying client request.

We use the diagram in figure 4 to describe the different transactions involved in satisfying a client request. To improve the clarity of the diagram we have not shown all the components of our system nor have we shown the queues associated with undeliverable messages. When the client issues a request the client handler forwards the client request, say M_1, to the local RRN (RRN_A). RRN_A starts a transaction T1 that is successfully completed when M_1 has been placed in the request queue by the RProvider. The process of moving M_1 to the initial target destination (RRN_B) is achieved by the RServer and is contained within T2. The RServer takes M_1 from the request queue of RRN_B and issues a request to the target Web Service and waits for a reply. Once a reply, say M_2, is received it is placed in the response queue. However, if M_1 is undeliverable then M_1 is placed in the retry queue. This process is performed within T3. The response listener takes M_2 from the response queue and places M_2 in the response queue of RRN_A within T4. The RListener starts T5 and takes M_2 from the response queue and returns the reply to the client.

In our system we assume that clients and Web Services are non-transactional objects. Therefore, we may assume that the failure of a client or Web Service may result in system inconsistencies. For example, if during T3 a message is successfully delivered to the target Web Service but timeout occurs before a reply is received then M_1 will be placed on the retry queue. However, the target Web Service may be processing M_1 (as it was successfully delivered but the target Web Service was slow returning a reply). RRN_B may reissue M_1 to the target Web Service resulting in an undesirable repeated processing of M_1. If the target Web Service participated as a transactional object within T3 then a timeout (as described previously) may result in an aborted transaction (T3) causing the rollback of the target Web Service state (removing any state changes the delivery of M_1 may have caused) allowing M_1 to be reissued later. This approach may be supported by implementations of WS-Atomic Transaction [4] and WS-Coordination [5] specifications.

As communications may span organizational boundaries we provide security features to ensure that messages sent between RRNs are genuine. A signed digest of the message that is to be sent between RRNs is created and included in the message as a JMS message property. The public key associated to the private key that is used to sign the digest is distributed to all other RRNs. This enables an RRN to verify the identity of the sender of a message: if signing a digest of the message contents with the public key identifies the same set of keys as signing the message with the private key, then the sender is genuine. This precaution provides security in the sense that the identity of a message sender as that of a known RRN. There is a measure of non-repudiation incorporated into such a communication as when an RRN signs a message and it is verified, the administrator of the signing RRN cannot later deny having ever sent the message.

4. Related Work

The work presented in this paper is an engineered solution to non-repudiation and reliability that may be adapted to fit associated Web Service standards. In this section we concentrate on how our system relates to such standards.

A specification exists that enables Web Services to participate in atomic transactions (WS-Atomic Transaction) [4]. As previously mentioned in 3.4, employing atomic transactions for client/server interactions with an RRN would make our system more robust. Furthermore, it may be possible to enhance our system with WS-Atomic Transactions to enable inter-RRN communications. However, allowing clients and servers to interact directly using WS-Atomic Transactions would have the drawback of presenting a tightly coupled environment where transient unavailability of transaction participants would result in the aborting of transactions (a scenario our system attempts to overcome). Furthermore, transactions are a heavyweight process (requiring all participants to carry out two phase commit protocol) and it is unlikely that every RPC would need to be carried out as an atomic transaction. The use of transactions would also inhibit the ability of a client to be released from RPC interaction to continue processing and return at a later time to receive a reply (see 3.3). To implement such a scenario will require more long lived transactions that employ compensation techniques [9], but this approach in itself does not satisfy non-repudiation requirements.

The nature of the implementation of a WS-Transaction service has to be considered in relation to our non-repudiation approach. The coordinator is responsible for determining the outcome of a transaction and is provided by the WS-Coordination service [5]. This makes the coordinator role crucial to the outcome of transactions with the need to ensure all transaction participants trust the coordinator. However, the coordinator must take part in our message logging scheme for non-repudiation to provide similar functionality to our system.

Confluent Software developed its own CORE Web Services Monitoring and Management Platform [8] (which now forms part of Oracle's Identity and SOA Management solutions framework [18]). The purpose of the platform is to allow an organization to implement Service-Oriented Architectures while offering full control over how a service is deployed and executed. Policies that govern how such a service operates may also be described and include Quality of Service, security and message logging. The focus of the CORE platform is on security and logging, although it does provide support for RPC. Our approach is different as we apply a MOM oriented solution.

Work carried out by Maheshwari et al [17] and Tai et al [16] specifically describes a system which enhances Web Service reliability. These works are interesting as MOM is highlighted as a suitable mechanism for implementing underlying reliability for Web Services. Similar observations to our own are made in these papers: loosely coupled MOM architecture is an ideal candidate for underlying messaging infrastructure implementation for Web Services. However, these works do not address the non-repudiation element which we ourselves see as an integral part in any inter-organizational function. However, the reliability element is extensively researched in these papers, with QoS parameters described and testing provided.

5. Conclusions and Future Work

We have developed a system that provides reliable messaging across organizational boundaries while implementing suitable mechanisms for non-repudiation for clients and servers that use SOAP RPC to interact and WSDL to describe services. We have tackled the problem by using a novel approach of employing MOM technologies to achieve inter-organizational communications. By using MOM, the loosely coupled association between sender and receiver has been exploited to prevent limited transient client/server unavailability from hindering successful completion of an RPC. Furthermore, the persistent messaging and transactional services available to MOM technologies ensure that partial failure of our system does not necessarily result in loss of messages.

Our system is built in a modular fashion. We are in the process of tailoring our services so that they adhere more closely to Web Service standards that dictate how non-repudiation and reliability may be utilized.

References

[1] Sun Microsystems, “Java Message Service. Version 1.1, April 12, 2002”, http://java.sun.com/products/jms/docs.html as viewed January 2004

[2] D. Ingham, Arjuna Technologies Limited, “ArjunaMS Documentation”, http://www.arjuna.com/products/arjunams/index.html as viewed January 2004

[3] Apache Web Services Project, “The Axis Toolkit, version 1.1”, http://ws.apache.org/axis/ as viewed January 2004

[4] Arjuna Technologies, Ltd., BEA Systems, Hitachi, Ltd., International
Business Machines Corporation, IONA Technologies, Microsoft Corporation, Inc., “Web Services Atomic Transaction (WS-Atomic Transaction)”,
http://www-128.ibm.com/developerworks/library/specification/ws-tx/, as viewed December 2006

[5] Arjuna Technologies, Ltd., BEA Systems, Hitachi, Ltd., International
Business Machines Corporation, IONA Technologies, Microsoft Corporation, “Web Services Coordination (WS-Coordination) Specification”, http://www-128.ibm.com/developerworks/library/specification/ws-tx/, as viewed December 2006

[6] OMG, “Notification Service Specification”, OMG TC Document telecom/99/07/01, 2000.

[7] The World Wide Web Consortium (W3C), “Simple Object Access Protocol (SOAP) (version 1.1)”, W3C Note 08, May 2000

[8] Confluent Software Inc., “Confluent Software Inc Solutions”, http://www.confluentsoftware.com/solutions, as viewed September 2003.

[9] K. Gottschalt et al., “Introduction to Web Services Architecture”, IBM Systems Journal, Vol 42, No 2, 2002

[10] Object Management Group, “The Common Object Request Broker: Architecture and Specification, 2.3 edition”, OMG Technical Committee Document formal/98-12-01, June 1999

[11] A. Gokhale et al., “Reinventing the Wheel? CORBA vs. Web Services”, WWW2002, The Eleventh International World Wide Web Conference, Honolulu, Hawaii, USA, 7 – 11 May 2002

[12] The World Wide Web Consortium (W3C), “Web Services Description Language (WSDL) (version 1.1)”, W3C Note 15, March 2001

[13] The World Wide Web Consortium (W3C), “Extensible Markup Language (XML) 1.0 (second edition), W3C Recommendation 6 October 2000

[14] Akamai Technologies, Computer Associates International, Inc., Fujitsu Limited, Hewlett-Packard Development Company, International Business Machines Corporation, SAP AG, Sonic Software Corporation, The University of Chicago and Tibco Software Inc., “Web Service Notification (WS Notification) and associated specifications”, http://www-128.ibm.com/developerworks/library/specification/ws-notification, as viewed December 2006.

[15] BEA Systems, IBM, Microsoft Corporation, Inc, and TIBCO Software Inc., “Web Services Reliable Messaging Protocol (WS-ReliableMessaging)”, http://www-128.ibm.com/developerworks/library/specification/ws-rm/, as viewed December 2006

[16] S. Tai, A. Mikalsen, I. Rouvellou, “Using Message Oriented Middleware for Repiable Web Services”, Web Services, E-Business, and the Semantic Web, Springer Berlin / Heidelberg LNCS, Volume 3095/2004, pp 89-104, July 2004

[17] P. Maheshwari, H. Tang, R. Liang, “Enhancing Web Services with Message-Oriented Middleware”, Proc. IEEE International Conference on Web Services (ICWS’04), 2004

[18] Oracle Corporation, “Oracle Fusion Middleware”, http://www.oracle.com/products/middleware/index.html, as viewed December 2006