A Modular Architecture for Secure and Reliable Distributed Communication

C. M. Jayalath, R.U. Fernando

Abstract—**Over the past decade various efforts were taken to provide mechanisms to do secure and reliable message exchanges in distributed systems. With the advance of the Internet and concept of SOA much focus and effort were given to realizing this based on Web services. Our goal was to come up with a solution which implements these features in a usable and a modular manner. The implementation was done on top of the Apache Axis2 platform and the result was a framework which facilitate fully secure and reliable Web service message exchange.**

Index Terms—**Rampart, SOA, Sandesha2, Web Services**

I. Introduction

Reliability and Security are two of the most important aspects for any distributed communication mechanism. With security the expectation is to address several aspects including secret communication, digital signing and avoiding intentional message repeats from a possible attacker. With reliability the idea is to make the communication guaranteed. Most of the modern communication applications need one or both of these aspects. Because of this many of the standards that appeared in the field of communication were incorporated with ways to address these two issues or they were later extended to address those.

The field of Web Services was no different. Initial web service specs SOAP, WSDL and UDDI simply addressed the basic needs of the technology namely mechanisms for describing message protocol, service description and service discovery. But later when the standards came into practice it was understood that this would not be enough. This caused the beginning of a full stack of WS-* specifications. One very important part of this stack was to provide means for the above two aspects, security and reliability.

A. Previous Work

There have been quite a number of efforts from various vendors and institutions to develop security and reliability for distributed systems. Some of these have been proprietary solutions. Some were open source. Some of the implementations came from academia.

Microsoft WCF (Windows Communication foundation)[1] is one of the leading efforts being taken to provide a solution in this space. With WCF Microsoft tries to build a unified distributed communication platform totally based on the Web service stack. Because of being based on Web services they will easily be able to communicate with other distributed technologies unlike their previous attempts like COM+ or .Net remoting. WCF implements popular Web service specifications for doing secure reliable message exchanges.

Sun Microsystems is trying a different path with their J2EE family of specifications. They introduced several specifications to address various aspects of the Web service arena including JAX-WS (for web service communication), JAXB (for XML data binding). JAXP (for XML processing) and SAAJ (for attachments). Many J2EE implementations have implemented these specifications.

Some of these implementations are trying to provide means to do secure, reliable communication by implementing popular Web service specifications such as WS-ReliableMessaging and WS-Security.

There has been several efforts from the academia as well. One family of solutions [2] from the Cornell University, Computer Science department is to provide means to do reliable communication in time critical environments. Another effort [3] from Indiana University, Computer Science department is on building a broker based platform to provide ways to do distributed communication with secure and reliable aspects. More details on these solutions will be given later.

B. Our approach

The approach that was analyzed and implemented by us was based Apache Axis2 the latest and the most promising Web service stack from Apache software foundation. Apache Axis2 has an architecture that reflects the changes that happened in the Web service arena in the past few years. It also gives features to easily embed implementations of newly introduced specifications into the framework. Extensions which introduces this kind of new functionalities into Axis2 are called modules.

Two solutions were developed by us based on Axis2. First one was for providing security needs. This consisted of two Axis2

C.M. Jayalath is with Web services project of the Apache Software Foundation. (e-mail:chamikara@apache.org)

R.U. Fernando is with the Web services project of the Apache Software Foundation (e-mail:ruchithf@apache.org)

modules named Rampart and Rahas. Rampart provides basic security needs such as signing and encryption and also provided means to utilize WS-SecureConversation. Rahas was a implementation of WS-Trust. Mode details on these specifications will be given later.

Another solution named Sandesha2 was aiming at providing the reliability need for Axis2. This achieved acknowledgment based reliability by implementing the WS-ReliableMessaging specification. Sandesha2 also gives an in-order exactly-once delivery assurance and support for persistent storage based reliability giving a much higher value in real business scenarios.

The design and implementation of the systems were done after the careful consideration of several features that could give a much higher value to the end user.

1)Pluggability:

It is very important that the framework provides flexibility in its implementation to the users. The flexibility was achieved by decoupling as much as possible. Using the module approach in Apache Axis2 independent modules were provided that catered for the security aspects and reliable messaging aspects. These two can independently act on messages and when both are available in the system they will be able to perform secure-reliable messaging.

2)Versioning:

Interoperability being the most important goal of Web services and functionality extension specifications it is very important that the developed framework supports the latest released specifications and that it is capable of extending itself to support inevitable changes in the revised specifications. In implementing these quality of service specifications Apache Axis2's ability to support versioned modules is very important.

3)Configuration:

Deploying a secure reliable messaging with a customer security requirements can be a nuisance when it comes to configurations. Therefore the proposed framework uses standard domain specific policies for both security and reliable messaging. Compared to having a custom configuration language where it expects the users to understand security and reliability requirements and come up with appropriate configurations, this is far more practical and convenient to users.

C.Summary of the paper

This paper will cover the basic design approach and implementation approach that were taken by us when building a security and reliability layers for Apache Axis2. It will start by giving a general introduction to Web services. Then it will move to the more interesting part on the design and implementation of our system. The latter parts will introduce the results that were obtained by analyzing several other similar implementations and comparing them with our solution.

II.Background

A.Introduction to web services and ws-*

Web services is a XML based inter application communication mechanism. The definition for the Web Services given by W3C WS Architecture working group is given below.

[Definition: A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.][4]

Even though this definition gives a very abstract view, practically Web Services are based on three basic specifications namely SOAP, WSDL and UDDI [4]. All three of these specifications have been well established and finalized versions have been released under W3C.

SOAP is a extensible XML based messaging protocol which defines the wire format of a message interaction between a client and a Web Service. Extensibility of SOAP paves the way for easy adoption of other protocols into the stack.

WSDL is the primary and most widely used description language for Web Services. This defines a XML based language to define various aspects of the services including service operations, formats of the input and output message into these operations, way the service binds into various protocols and the actual endpoints of these bindings.

UDDI is the widely adopted protocol for discovering the Web Services. This also gives a registry facility to support the process of publishing and discovering services.

On top of these three basic specifications a set of other Web Service specifications were developed mainly by groups of software vendors. Some of these specifications which got a good momentum were later submitted to the two standers bodies W3C or OASIS. Some of these have been released but most are still under development. However this led to a stack of protocols each adding a useful feature to the space of Web services.

B.Web Service specs for secure reliable communication

1)WS-ReliableMessaging

WS-ReliableMessaging is the most prominent Web service specification for reliable communication. The specification was originally initiated by a collaborative effort of Microsoft, IBM, BEA and TIBCO. This version of the specification was called 1.0. Later this was submitted to OASIS for standardization and version 1.1 of the specification is being developed under the OASIS WSRX technical committee.

WS-ReliableMessaging specification basically tries to obtain reliability trough a simple acknowledgment based mechanism. Message exchange will always happen within a context named a sequence. Two entities hoping to do a reliable message exchange will first have to establish a sequence by exchanging several protocol specific messages. After establishing of this client will send application messages to the server and the server will send acknowledgment messages time to time. Each application message in a sequence will be numbered with a unique message number. Server can use this to do an ordered invocation of messages.

2)WS-Security, WS-Trust and WS-SecureConversation

These are the family of specifications facilitating the security needs of the Web service space. WS-Security is aimed at providing mechanisms to do secure SOAP message exchanges. WS-Security enhances the basic SOAP model by providing means to guarantee message integrity, message confidentiality and authentication of messages. The specification also provides an extensible mechanism to associate various security tokens with messages.

WS-Trust specification provides extensions to the WS-Security specification by providing ways to issue, exchange and validate security tokens. This also facilitates issuing and distribution of security credentials within different trust domains.

WS-SecureConversaiton is built on top of both the WS-Security and WS-Trust specifications. to provide mechanisms to to secure communication between services. The main focus is on describing ways to establish security contexts and to do secured message exchanges within these contexts.

3)WS policy

WS-Policy gives a model to describe and exchange the policies of web services. WS-PolicyAssertions specification provide ways to express capabilities and constrains of a certain Web service while WS-PolicyAttachement define several ways to attach these policies with Web services.

Other specifications uses the policy specifications described above or extends it to provide means to specify policies of that particular domain. For example the WS-SecurityPolicy specification provides these features for the security set of specifications described above.

III. Architecture

A. Axis2 architecture

Apache Axis2 has quite a set of architectural concepts is not possible to be covered here. Mainly Axis2 consists of several subsystems as given in the diagram below.

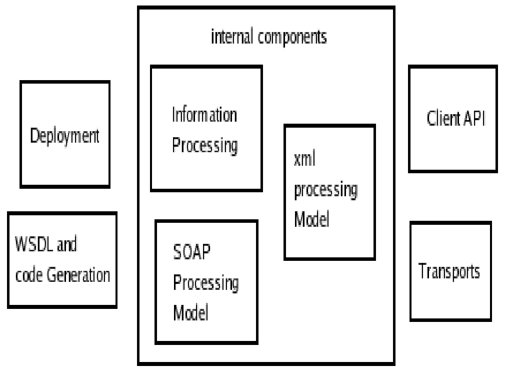

Fig. 1: Architecture of Axis2

The XML processing model of Axis2 is based on a StAX based XML representation named AXIOM. AXIOM provides features like differed building which optimizes the performance by keeping the data in the transport stream until they are really required.

The SOAP processing model of Axis2 consists of a set of Flows each containing of a set of handlers which will do various operations on the messages passed through them. The handlers are organized in logical groups known as Phases. The transport model is responsible for accepting the SOAP requests from various transport mechanisms and for delivering them to the SOAP processing framework correctly. It is responsible for writing response SOAP message back to the respective transports as well.

The deployment model gives facilities to do achieve based deployment of services and modules. The descriptor files contained within these archives will give exact information on how these archives should be deployed. Modules are the mechanism that have to be used to extend the functionality of Axis2. A module may introduce a set of handlers which may need be added to the various flows within the Axis2 system.

B. Our solution for reliability

Sandesha2 was the name of our solution for reliable communication. Sandesha2 was also a module built on top of Axis2. So it was leveraged from all the features that were readily available there.

Sandesha2 implemented the WS-ReliableMessasing specification that was described above. It currently supports both available versions of this specification. Therefore Sandesha2 basically uses an acknowledgment based model to reliably deliver of SOAP messages from one endpoint to another.

Apart from the basic functionalities that were available by implementing the specification it was taken a bit ahead by adding several other features that could increase its value quite a lot in a real business scenario.

One of those was the delivery assurances provided by Sandesha2. It supports InOrder Exactly-Once delivery assurance. So Sandesha2 can guarantee that your messages are delivered to the server endpoint in the same order they were sent at the client side. Also it guarantees that non of your messages will be delivered twice to the server endpoint. Ordering is optional and could be disabled. One reason for this may be the performance.

Another feature Sandesha2 provide is the support for persistent storage. This could be really valuable in real business scenarios both for the server side and the client side.

In the server side RM state will be preserver in crashes. Assume when the server crashed it had hundred ongoing RM sequences with various clients. These could be in various stages. When the system comes back all the RM data will be restored. Server will start transmitting and retransmitting pending messages and will start performing pending invocations. Clients will be able to interacted with the server from the point they were when the system crashed.

The real value for persistence will come in the client side. This gives a very high level of guarantee to the client and promises the delivery of messages even in client crashes. Sandsha2 will simply start the sequence from the place it was last saved will will start reliable delivery of messages to the other side.

Sandesha2 follows a WS-Policy based configuration model. Because of this, Sandesha2 configuration parameters can easily be presented to the outside using standard mechanisms like WSDL. Policies can be in the module itself (which are the default values), in a service or in a operation. When policies are present in multiple levels affective value will be picked from the lowest level. For example if module and service define two values for the RetransmissionInterval policy (which gives the interval on which the messages are retransmitted) for thatparticular service Sandesha2 will pick the value defined there. But for other services value mentioned in the Module will be used.

Following diagram gives the architecture of Sandesha2.

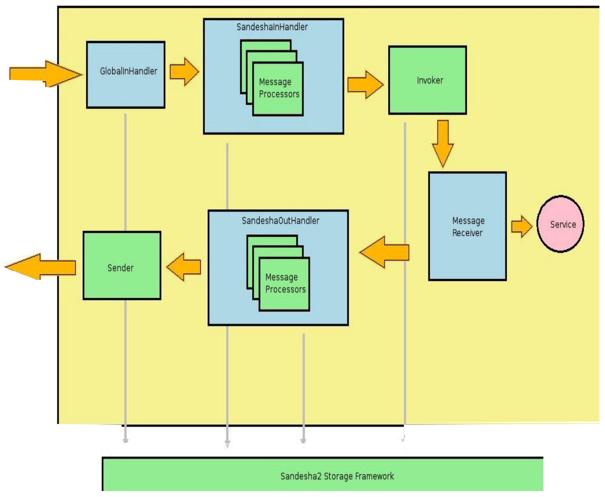

Fig. 2Architecture of Sandesha2

As it is given in the diagram Sandesha2 introduces three Handlers to the execution chain of Axis2. These Handlers will delegate the processing of incoming and outgoing RM messages to a set of MessageProcessors. Each MessageProcessor is responsible for processing a specific type of RM messages. For example CreateSequenceMessageProcessor will process CreateSequence messages.

Sender and Invoker and two Thread pools present within Sandesha2. Sender is responsible for the transmission and retransmission of messages while the Invoker is responsible for the invocation of messages in order to guarantee the ordered delivery feature.

Sandesha Storage framework defines a set of interfaces that could be implemented by a particular storage mechanism. These interfaces define several beans (which could represent database rows in a OR mapping) a set of BeanManager (with CRUD methods to manipulate these beans) and a transaction layer. The rest of the Sandesha2 code completely runs on top of this set of interfaces defined by the Storage framework. Because of this you can easily define a storage framework for your own storage mechanism and make sandesha2 work on top of that.

C.Our solution for security

Apache Rampart was designed specifically to support WS-Security and WS-secureConversation using WS-SecurityPolicy as the main configuration language.

1)Rampart Axis2 Module

Rampart was developed as an Axis2 module which is packaged as a .mar file and could be dropped into an Apache Axis2 repository. When the module is available in an Axis2 repository it can be engaged at service or operation level. The module will not process messages unless it is configured.

This module consists of two handlers:
RampartSender
RampartReceiver

It is very important to note the positioning of these two handlers in the execution chains – outflow and inflow. RampartSender is placed in the "Security" phase in the outflow after the message out phase.

This is critical to ensure that RampartSender is the last handler that will modify the message before it is written to the wire.

Similarly RampartReceiver is placed in the "Security" phase of the inflow which placed right after the transport phase. In this case the position of the handler is important due to several reasons. First a service must be configured to apply and enforce security on messages directed towards it. Therefore the service or operation must be discovered to pick up the relevant security configuration to enforce security policies on the incoming message. Therefore it is important that "Transport Dispatchers" in the "Transport Pahse" discovers (or dispatches) the service and/or the operation. Furthermore any part of the message should not be processed before message integrity is verified. Therefore all other handlers that uses information from the incoming SOAP message must be positioned after RampartReceiver.

It should be noted that Rampart Axis2 module is configured according to the WS-Policy framework using WS-SecurityPolicy and some Rampart specific assertions. The Rampart specific assertions provides additional meta-information required in enforcing policy specified by the WS-SecurityPolicy assertions.

2)Rahas – WS-Trust components

Rahas provides the SecurityTokenService (STS) functionality and a client API required to interact with the STS according to the WS-Trust specification. The STS functionality provided by Rahas comes in two flavors, a service and a module. The STS service is an Axis2 web service that can act as a standalone STS. And the STS module is an Axis2 module which can be engaged on an existing service. This module will append the additional operations into the service to be able to handle different types of security token requests.

The Request Dispatcher figures out the type of request coming in and then routes it into the configured Token Issuer, Token Canceler, Token Renewer or Token Validator implementation. All request information include intermediate processing results are stored in the Rahas Data data structure and this is used by the implementations to obtain required information on the request. The STS will be configured with a Token Storage implementation and this will be used by the Token Issuers, Token Cancelers, Token Renewers and Token Validators to store and obtain security tokens.

3)Rampart and Rahas Marriage

In supporting WS-SecureConversation scenarios the initial handshake requires the service to be aware of the WS-Trust protocols. Therefore the Rahas module (STS module) is used to append STS operations to the service. The initial handshake is secured by the bootstrap policy specified in the policy in establishing the security context token and Rampart handlers tracks the establishment of the security context and uses the Token Storage to hold the security context token.

D.Combining Security & Reliability

At first there was no need for the combination of the two modules. Depending on the configuration set by the user Rampart/Rahas could guarantee the secured message exchange and Sandesha2 could guarantee the reliable ordered delivery of messages.

But with the advances of the underlying protocols there was a need to bring the ReliableMessasing and SecureConversation contexts together. Each Reliable Messasing sequence had to have a associated SecureConversation session. At the destination Reliable Messaging layer had to validate the messages to make sure that the each message of a sequence processes the correct security tokens.

To leverage this a SecurityManager interface was introduced to Axis2. An interface based model was followed to minimize coupling and to allow a future integration of a different SecureConversation implementation into Sandesha2. The SecurityManager introduced several functions to do tasks such as manipulating issued security tokens, validating RM messages and attaching security tokens into RM messages. An implementation of this named RampartBasedSecurityManager was developed by us to leverage the marriage of Sandesha2 and Rampart implementations.

IV.Implementation

A.Implementing Reliability

Sandesha2 project was started on late 2005 and a 1.0 release was done on May 2006.

The implementation is being done in two languages. The first one was the Java implementation which introduced most of its concepts and is to be used with the Axis2 Java implementation. This was quickly followed by a C implementation aiming the Axis2 C stack. The C implementation tries to provide reliability with a maximum level of performance because of being run in a native (non virtual machine) environment.

B.Implementing Security

The implementation of Rampart uses Apache WSS4J for producing and processing secured SOAP messages. Apache WSS4J uses XML-Security to obtain XML-Signature and XML-Encryption functionality.

In securing messages WSS4J provides a set of message builders to perform different security operations on a SOAP message such as addition of a timestamp and/or a username token into a security header, encrypting a part of a message, signing a part of a message. These are used by the Rampart Sender handler in constructing the secured SOAP message as specified by policy.

In processing a secured message Rampart Receiver handler hands over control to the Security Engine provided by WSS4J. This processes the security header of the given SOAP message and performs required operations to authenticate, validate and decipher the message. Once this is completed Rampart Receiver will process the results of security processing against the policy to check conformance with the policy.

V.Analysis/Comparison

A.Comparison with vendor specific implementations

Today it is possible to find a vast number of SOA solutions provided by various vendors. These solutions are mainly driven by various software firms and are taking different approaches in the approach to develop a secure and reliable distributed communication mechanism. The approach that were taken by us had several advantages over the approach that were taken by these vendor driven solutions. Some of them are listed below. Our main focus was on popular SOA stacks such as WCF from Microsoft and J2EE from Sun Microsystems.

1)Loosely coupled components

As it was stated above the main integration of security and reliability components happens through a SecurityManager interface. Someone who wants to combine his own SecureConversation implementation with Sandesha2 module can easily do so by defining his own SecurityManager implementation. Also there is nothing that prevents somebody who want to use his own RM implementation with Rampart and Rahas. Many other systems available in the marked do not show such composability. In most of the case the components in their systems are tightly coupled and it is far too difficult or impossible to plug-in a part of an different implementation into it.

2)Simple and SOA oriented API

Many web service framework vendors provide easy to use API for making the task easy for web service developers. Most of these APIs resembles the previous languages or programming methodologies provided by the same vendor. Because of this the many APIs have lost the focus on SOA and the final solution have become quite difficult for a client who is not familiar with similar technologies from the same vendor. Due to this complexity many APIs had lost their SOAness. What was meant by this was the easiness for a user who is not familiar with the particular technology but who is familiar with general SOA concepts to adapt to and use these APIs.

Our solution uses the simple API based on the ServiceClient, OperationClient based approach of Axis2. In Axis2 a ServiceClient simply represents a client for a remote service. A user can ask the ServiceClient for OperationClients, each representing a operation of this service . All the parameters to the ServiceClient are passed through an Options object which has get/set methods for manipulating commonly used values and a property layer to enter any other additional parameters.

Both Sandesha2 and Rampart are based on this simple API. No additional extensions are added other than introducing some keys for users to enter certain domain specific properties. This makes the API much simple an easy to use for any developer familiar with general SOA concepts.

3)No proprietary protocols

Most of the frameworks provide support for the open specifications but many of these provide additional support when both the client and service implementations are from the same vendor. Some of these extensions leverage the features enabled from specifications like WS-Policy but there is no guarantee to say that the other vendor specific protocol mechanisms will not be used.

This creates several problems. Firstly this will confuse the user. He will have to bypass the general mechanisms offered by service description languages such as WSDL to find out other information about the service, such as the vendor and the software version. Secondly this will seriously harm the interoperability. The client will be able to nicely interoperate with a service developed using the same technology but may have trouble as soon as he tries to access a service developed using a different one. This will confuse the users and will undermine the whole idea of using open standards which is having complete interoperability between implementations from various vendors.

Our solution is completely open and do not use any closed communication mechanisms. Our solution is fully adherent to the open standards provided in the Web service stack. Any extension was and will be based on open and excepted mechanisms such as WS-Policy.

4)Not a mix of legacy and SOA

Many of the solutions provided above tries to maintain backward compatibility with other legacy distributed computing mechanisms, most of the time from the same vendor. The reasons are mainly financial or marketing not merely due to the technological requirements or limitations of the newly introduced technology.

Due to the fact of trying to being compatible with these legacy and mostly outdated technologies some of the vendors lack the opportunity to freely architecture the new product to fully leverage the benefits offered by SOA. They may end up in having links to old concepts like the distributed object model and may lack both in the cleanness of the architecture and in the overall feature set offered to the user. This may also affect the API which now may be a more generalized version than a one that tries to resemble the concepts of SOA.

Our implementation completely focuses on providing a Web service solution. There was no aim to have a goal to come up with a solution generalized with other distributed communication mechanisms and there will not be such an aim in the future as well. The soul aim was in coming up with a architecturally clean solution to realize the Web service stack and provide a user friendly API which resembles the values offered by SOA.

B.Comparison with solutions from academia

This section will present two distributed secure/reliable systems that were developed in academia and will try to analyze and compare the approaches that were taken by them with the our solution.

1)Ricochet from Cornell University

A family of solutions from the Cornell university is aimed at providing a time-critical and reliable experience to the real time needs of the distributed systems. The work is based on a protocol called Ricochet.

Ricochet is aimed at an environment where a large number of nodes will decide each one belonging to one or more groups. Ricochet obtains reliability by using error correction mechanisms to recover lost packets. At the receivers an error correction packet will be generated by using a random set of groups to which that particular node belongs to. This error correction packet will be send to the other nodes that belong to the same set of nodes. Nodes use the incoming error correction packets to recover lost packets or burst losses.

For the packets that cannot be recovered Ricochet uses an acknowledgment based model. The receivers sends negative acknowledgments to the senders asking for a retransmission of the packets that could not be recovered.

There are several differences between the attempt followed by the Ricochet team and the approach that were followed by us in the space of reliability in distributed systems. Ricochet was more aimed at IP level using IP multicast techniques to send packets back and forth where as were were aiming at the application layer protocols. Ricochet followed a mix model of error correction and acknowledgments but our system was totally based on an acknowledgment/retransmission based model. Overall the solutions are aimed at two environments. Ricochet aims at a datacenter kind of environment where a large number of nodes communication with each other where both reliability and time criticality is a requirement. Our aim is at the reliable communication between two entities through the Web where reliability is the main concern.

2)Narada broker from Indiana University

This is a solution based on the concepts of brokering. A Narada system will consist of a set of widely dispersed set of brokers. These brokers are organized in a hierarchy where a brokers consists of a cluster which is a part of s super-cluster etcetera. Communication between a cluster is generally quite efficient and reliable than communication between the same cluster and a super-cluster.

Clients register their interests and message formats in the broker system. After this they can freely disconnect and the system will make sure that that events which map to these requirements are flown and delivered to the client in subsequent reconnects. Clients will normally connect to their local broker than to a remote one minimizing bandwidth limitations that could occur if a large number of remote clients connect to the same remote broker.

Developers of this system have implemented distributed protocols like JMS and the Web services stack on top of this brokering environment. An implementation of WS-ReliableMessaging has been developed to facilitate reliable delivery of messages. Nadada has its own subsystem to manage secured message exchange.

When comparing this with our solution again a difference in the environment where these two will be applied is visible.

Narada is more aimed at a widespread network of computers which are primarily doing the communication using a underlying brokering system. Even though our approach was based on similar specification the targeted environment is basically the Internet.

VI.Future Work

The work was done by us to bring security and reliability layers into Axis2 is not over. There are several areas that could be improved or modified and some other where one more research could be carried out.

One key aspect is performance. It has been noted that when both Sandesha2 and Rampart are acting together there is a noticeable reduction in performance specially for messages of larger sizes. One reason for this could be the number of protocol messages that are exchanged in Sandesha2. Another could be the DOM conversion that happens in Rampart.

Sandesha2 send a number of protocol message back and forth thought the lifetime of the sequence. Some of these may be sent only once per sequence but some others like Acknowledgment messages get transmitted quite often. To minimize traffic several features were introduced to optimize this flow of acknowledgments. One way was piggybacking acknowledgment messages with other application messages that are aimed at the same destination. Another way was to delay sending of these messages until a certain time period expires. Still more could work be done in this area. More ways have to be found out to optimize the acknowledgment flow. This could give a considerable improvement to the overall performance of the system.

Axis2 is based on a StAX based XML infoset representation called AXIOM. But WSS4J and XML-Security uses DOM as the XML object model. To bridge this gap the authors came up with a hybrid implementation of DOM and AXIOM called DOOM. When a SOAP message reaches a Rampart handler is first converted into DOOM and then the components that expects the input to be DOM processes the message using DOM interfaces and the components that expects the message to be AXIOM uses AXIOM interfaces. However this conversion proves to be very expensive and it forgoes one of the main the advantages of using AXIOM, which is deferred building.

Another key area that more work have to be on is interoperability. Some work was done to interoperate our system with other popular systems from vendors like Microsoft, IBM, SAP and Oracle but the work is not over. Some of the scenarios interoperated quite well but more work are needed in several others. Yet these interops were quite promising and the successfully completed scenarios showed that the goals explained in the specifications were truly realizable.

VII.Conclusion

Our aim was on providing a framework that would enable secure and reliable message exchanges in distributed systems. Apache Axis2 was chosen as the underlying framework and several leading Web service specifications were implemented to provide mechanisms to do secure and reliable message exchanges. The end result was consisting of modular components which could be used separately or could be combined to do a fully secure and reliable exchange of messages within a distributed system.

References

[1] (2006, Mar.). Windows Communication Foundation Architecture Overview. Microsoft Corporation. [Online]. Available: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnlong/html/wcfarch.asp

[2] M. Balakrishnan and K. Birman, "Reliable Multicast for Time-Critical Systems,"

[3] G. Fox and S. Pallickara, "The Narada Event Brokering System: Overview and Extensions,"

[4] (2004, Feb.). Web Services Architecture. W3C. [Online]. Available: http://www.w3.org/TR/ws-arch/

[5] V. Tosic, A.V. Moorsel and R. Wong, "Quality of Service (QoS) Middleware for Web Services," In MWS 2005.

[6] A. Sheth, J. Cardoso, J. Miller and K. Kochut, "QoS for Service-oriented Middleware," In Conference on Systemics, Cybernetics and Informatics, Orlando, FL, July 2002.

[7] K. P. BIRMAN and THOMAS A. JOSEPH, "Reliable Communication in the Presence of Failures," In ACM Transactions on Computer Systems (TOCS), Feb 1987

[8] X. Liu, C. Kreitz, R. van Renesse, J. Hickey, M. Hayden, K. Birman and R. Constable, "Building reliable, high-performance communication systems from components," In ACM Symposium on Operating Systems Principles, 1999

[9] D. F. Ferguson, B. Lovering, "Secure, Reliable, Transacted Web Services: Architecture and Composition,"

[10] K. P. Birman, "T H E PROCESS GROUP APPROACH TO RELIABLE DISTRIBUTED C O M P U T I N G,"

[11] S. Pallickara, G. Fox, B. Yildiz, S. L. Pallickara, S. Patel and D. Yemme, "On the Costs for Reliable Messaging in Web/Grid Service Environments,"

[12] S. Pallickara, M. Pierce, H. Gadgil, G. Fox, Y. Yan, Y. Huang, "A Framework for Secure End-to-End Delivery of Messages in Publish/Subscribe Systems,"

[13] K. Birman, M. Balakrishnan, D. Dolev, T. Marian, K. Ostrowski and A. Phanishayee, "Scalable Multicast Platforms for a New Generation of Robust Distributed Applications,"

[14] C. Vasters. (2006, jul.). Introduction to Reliable Messaging with the Windows Communication Foundation . Microsoft Corporation. [Online]. Available: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnlong/html/introtowcfreliablemessaging.asp