Chapter 2

Literature Review

Literature review of this project spans across many areas. One of the obvious areas is the web services standards and related specifications. Web services primarily use SOAP[3] as the messaging format. WS-Addressing[10] provides a way to address end point references in a transport independent way. WS-Reliable messaging specification[8] uses WS-Addressing[10] to correlate the request and response messages.

There are some set of standard protocols and standards to generally support transactions and messaging. 2PC[7] is the widely used protocol to achieve distributed transactions. X/Open promotes standards for many protocols to improve the interoperability. X/Open distributed transaction specification[11] standardize the use of 2PC protocol. JTA/JTS[12][13] provide java specific APIs for distributed transactions.


Queued transaction processing is used for processing a transaction between a client and an application server asynchronously in a distributed transaction processing environment having at least one transaction queue manager.
IBM has done some work related to this area. This includes their classification of various ways to integrate the web services and transactions. Http is an effort to build a reliable protocol on top of Http.

Finally there have been many researches for message oriented and object oriented transactions. Further these researches have been extended to middleware mediated transactions which combines the above two concepts to achieve better transaction support.

2.1 Web service standards

2.1.1 SOAP

Simple Object Access Protocol (SOAP) is a protocol to exchange information in a decentralized, distributed environments developed by Microsoft and IBM. SOAP can support to enable remote procedure calls (RPC) over HTTP using XML. SOAP protocol specification mainly consists of three parts.

1. SOAP Envelope

SOAP envelope describes what is in the message and how to process it. A SOAP envelope has a required body part which is used to send the actual message, and header parts which can be used to provide the soap envelope processing instructions.

2. Set of encoding rules

There are a set of encoding rules which specify how to encode application-defined data types into XML format. This is important since SOAP provides an inter operable XML based messaging format.

3. Convention to represent remote procedure calls and responses

SOAP defines a way to encode a RPC invocation request and the response into a SOAP envelope. This is used in RPC type service invocations.

2.1.2 WS-Addressing

Web services can be accessed by sending SOAP messages to their respective endpoints. However the endpoint details may depend on the transport protocol. And also there are some information required by the messaging systems in order to dispatch messages to corresponding processes and correlate them.

Web Services Addressing (WS-Addressing) defines two inter operable constructs that convey information that is typically provided by transport protocols and messaging systems. These constructs normalize this underlying information into a uniform format that can be processed independently of transport or application.
1. Endpoint references

A Web service endpoint is an entity where Web service messages can be targeted. Endpoint references convey the information needed to identify/reference a Web service endpoint. Endpoint references are suitable for conveying the information needed to access a Web service endpoint, but are also used to provide addresses for individual messages sent to and from Web services.

2. Message information headers

This defines a family of message information headers that allows uniform addressing of messages independent of underlying transport. These message information headers convey end-to-end message characteristics including addressing for source and destination endpoints as well as message identity.

WS-Reliable messaging uses WS-addressing headers to specify endpoint addresses and convey message related information.

2.1.3 WS-Reliable messaging

Reliable message delivery is a common concept in message oriented communication. WS-ReliableMessaging specification[8] (WS-RM) describes a protocol that allows messages to be delivered reliably between distributed applications in the presence of network failures.

The protocol is described in this specification in a transport-independent manner allowing it to be implemented using different network technologies. To support interoperable Web services, a SOAP binding is defined within this specification.

The protocol defined in this specification depends upon other Web services specifications for the identification of service endpoint addresses and policies. This protocol does not talk about the delivery guarantees and persistence. However WS-RM implementations can provide persistence and delivery guarantees using the available protocol constructs.

WS-Reliable messaging is based on a reliable message model which is given below.
Following diagram shows the entities and events in a simple reliable message exchange. First, the Application Source sends a message for reliable delivery. The Reliable Messaging (RM) Source accepts the message and Transmits it one or more times. After receiving the message, the RM Destination acknowledges it. Finally, the RM Destination delivers the message to the Application Destination.
Following steps illustrates a typical set of messages passed in one RM sequence and how it provides fault tolerance. It uses a acknowledgment based retransmission similar to TCP.

1. The protocol preconditions are established. These include policy exchange, endpoint resolution, establishing trust.
2. The RM Source requests creation of a new Sequence.
3. The RM Destination creates a Sequence by returning a globally unique identifier.
4. The RM Source begins sending messages beginning with MessageNumber 1. In the figure the RM Source sends 3 messages.
5. Since the 3rd message is the last in this exchange, the RM Source includes a <LastMessage> token.
6. The 2nd message is lost in transit.
7. The RM Destination acknowledges receipt of message numbers 1 and 3 in response to the RM Source's <LastMessage> token.
8. The RM Source retransmits the 2nd message. This is a new message on the underlying transport, but since it has the same sequence identifier and message number so the RM Destination can recognize it as equivalent to the earlier message, in case both are received.
9. The RM Source includes an <AckRequested> element so the RM Destination will expedite an acknowledgment.
10. The RM Destination receives the second transmission of the message with MessageNumber 2 and acknowledges receipt of message numbers 1, 2, and 3 which carried the <LastMessage> token.
11. The RM Source receives this acknowledgment and sends a TerminateSequence message to the RM Destination indicating that the sequence is completed and reclaims any resources associated with the Sequence.
12. The RM Destination receives the TerminateSequence message indicating that the RM Source will not be sending any more messages, and reclaims any resources associated with the Sequence.
2.2 Transactions and messaging standards

2.2.1 2PC

Two phase commit protocol[7] is a protocol to support transactions in a distributed environment. In a distributed environment there are multiple participants. These multiple participants update multiple data sources. Two phase commit protocol ensure either these participants commit or abort atomically.

Two phase commit protocol is executed by a process called the coordinator process and other participant processes. As the name suggests two phase commit protocol has two phases called prepare phase and commit phase. Both of these participants' life cycles has been defined by the state transfer diagrams.

![Figure 2-3 Coordinator States](image-url)
Before the commit process starts, both coordinator and participants processes are at the initial state. Commit process starts when the initiator sends the commit message to the coordinator. Getting the commit message coordinator sends the prepare message to all the participants and moves to the prepared state and waits until all the responses come. When a participant receives a prepare message from the coordinator it sends the response as 'yes' and moves to prepared state if it is prepared to commit or sends the response as 'no' and moves to aborted state if it is not prepared to commit. Here if a participant sends a 'yes' response it can't later say it is not prepared to commit. Once all the participants sends their responses coordinator can decide either to commit the transaction or abort it. If there is at least one 'no' response coordinator have to decide to abort the transactions. After that coordinator tells its participants either to abort or commit and then moves to either commit or abort state. Once the participants gets the global commit or abort message from the coordinator it moves to the corresponding state and sends the acknowledgment back to the coordinator.

2.2.2 X/Open distributed transaction standards

X/Open is a independent, worldwide, open systems organization which supports implementation of open systems. In the context of the distributed transactions, X/Open has standardize the interface between the Transaction Manager and the Resource Manager in order to make them as open systems[11].

X/Open distributed transaction processing (DTP) model assumes three software components.
Application program specify the transaction boundaries and specifies the actions that constitute the transaction. Resource managers provides the resources which application program updates during a transaction. Transaction manager is the main component which assigns identifiers to transactions, monitor their progress and do the transaction completion or failure recovery.

Out of these interactions X/Open specification introduces a standard interface to communicate between the Transaction manager and the Resource managers. These interfaces are specified in C programming language.

2.2.3 JTA

Java transaction API specification[12] provides a set of java interfaces to support distributed transactions. It specifies the local Java interfaces between a transaction manager and the parties involved in a distributed transaction system. Following diagram shows the interfaces it defines and the relevant areas of those specifications.
UserTransaction interface provides the application the ability to control the transaction boundaries programmatically. The application can obtain user transaction and use begin and commit method to demarcate the transactions.

Transaction manager interface allows application server to control transaction boundaries. Transaction Manager allows users to begin and commit transactions associated with a thread.

Transaction interface allows operations to be performed on the transaction associated with target object. This interface can be used to:

1. Enlist the transactional resources in use by the application
2. Register for transaction synchronization callbacks
3. Commit or rollback the transaction

XAResource Interface provides a java mapping of the industry standard XA interface based on the X/Open Specification. This interface defines the contracts between the Resource Manager and the Transaction Manager in a distributed transaction processing (DTP) environment.
JTA specification defines five players which are involved in a distributed transaction services. Each of these players contribute to the distributed transaction processing system by implementing different sets of transaction API and functionalities.

1. A transaction Manager provides the services and management functions required to support transaction demarcation, transactional resource management, synchronization, and transaction context propagation.
2. A application server provides the infrastructure required to support the application runtime environment which includes transaction state management.
3. A resource manager provides the application access to resources.
4. User application which uses the transaction provided by the application server.
5. A communication resource manager supports transaction context propagation and access to the transaction service for incoming and outgoing requests.

2.3 WS-Transactions

WS-Transactions defined in three specifications. WS-Coordination defines a common framework to coordinate web services activities among different web services using different types of coordinating protocols.

2.3.1 WS-Cooordination

WS-Coordination[4] describes an extensible framework for providing protocols that coordinate the actions of distributed applications. Such coordination protocols are used to support a number of applications, including those that need to reach consistent agreement on the outcome of distributed activities.

The framework defined in this specification enables an application service to create a context needed to propagate an activity to other services and to register for coordination protocols. The framework enables usage of existing proprietary transaction processing systems while providing an inter operable mechanism to communicate.

The following diagram shows typical usage scenario of the WS-Coordination specification to coordinate the activities among different web services.
1. App1 sends a CreateCoordinationContext for coordination type Q, getting back a Context Ca that contains the activity identifier A1, the coordination type Q and an Endpoint Reference to CoordinatorA's Registration service RSA.
2. App1 then sends an application message to App2 containing the Context Ca.
3. App2 prefers CoordinatorB, so it uses CreateCoordinationContext with Ca as an input to interpose CoordinatorB. CoordinatorB creates its own CoordinationContext Cb that contains the same activity identifier and coordination type as Ca but with its own Registration service RSb.
4. App2 determines the coordination protocols supported by the coordination type Q and then Registers for a coordination protocol Y at CoordinatorB, exchanging Endpoint References for App2 and the protocol service Yb. This forms a logical connection between these Endpoint References that the protocol Y can use.
5. This registration causes CoordinatorB to forward the registration onto CoordinatorA's Registration service RSA, exchanging Endpoint References for Yb and the protocol service Ya. This forms a logical connection between these Endpoint References that the protocol Y can use.
2.3.2 WS-Atomic transactions

WS-Atomic transactions specification[5] defines an atomic transaction coordination type that can be used with the WS-Coordination specification. This specification describes such coordination type protocols which can be used with the short lived atomic transactions.

Completion

This protocol is used to communicate between the initiator and the coordinator. Initiator starts the commitment processing by sending a commit message. After that coordinator starts the volatile 2PC and proceed to durable 2PC. Then the final result is send to the initiator.

Two phase commit protocol

Two phase commit protocol is used to perform the atomic transaction among the participators. This protocol ensures all the participators comes to a final decision. There are two variations of this protocol.

1. Volatile two phase commit
   Used with the participators who use the volatile resources such as memory cache.

2. Durable two phase commit
   Use with the participators use the durable resources such as databases.

2.3.3 WS-BusinessActivity

Similar to WS-Atomic transactions specification this specification also defines coordination types and protocols to be used with WS-Coordination specification. These coordination types typically has to be used with the long running transactions. There are two coordination types and protocols has defined in this specification.

Coordination types

There are two coordination types have defined with this specification called atomic outcome and mixed outcome. In the atomic outcome coordination type all the participators either end up with end state or compensated state while in the mixed outcome mode participators and be end up within any state.

Coordination protocols

There are two types of coordination protocols defined with this specification called BusinessAgreementWithParticipantCompletion and BusinessAgreementWithCoordinatorCompletion. The former protocol initiation starts by the participant while for the latter it is started by the coordinator.
2.4 Queued Transaction processing

Figure 2-8 Queued Transaction Processing

Queued transaction processing is used to process transactions in a distributed environment asynchronously. This happens within three transaction boundaries. Firstly user application creates the request message and enqueues the request message to request queue within a transaction. After that server dequeues the message, process it and enqueues the response to response queue within another transaction. Finally user application dequeues the message from the response queue.

2.5 Different types of reliable web services

2.5.1 Using Message Oriented Middleware for Reliable Web Services Messaging.

Web services are applications that are described, published and accessed over the web using open XML standards. Different Message Oriented Middleware can be used with web services. Reliable communication is one of the most important aspects of any application. There are five ways that an web service can use MOM.

1. Messaging Middleware Reliability

Messaging middleware is specialized software that accepts messages from sending processes and delivers them to receiving processes. The two principle styles for MOM is centralized and distributed.
2. Aspects of reliability

The main aspect of the reliability is to tolerate the network failures. MOM can tolerate the network failures by repeatedly sending the message until it is acknowledged by the receivers component. In addition to acknowledged delivery, ordered delivery is another aspect of reliable messaging. Further important aspect of reliability is the integration of a message delivery in a larger processing context. Therefore a MOM should be able to group a message with other messages and other process activities.

2.5.2 Three facets of Reliability

1. Middleware endpoint to endpoint reliability

A message once delivered from an application to the messaging middleware, is guaranteed to be available for consumption by the receiving process.

2. Application to middleware reliability

The middleware's messaging API, supports reliability properties such as message delivery guarantees, message persistence and transactional messaging.

3. Application to application reliability

Sending and receiving applications engage in transactional business processes that rely on application-to-middleware reliability and middleware endpoint-to-endpoint reliability.

2.5.3 Reliable messaging for web services

This describes five different ways in which a web service can use the MOM for a reliable communication.

1. SOAP (with or without a reliability protocol like WS-ReliableMessaging) is used with an unreliable transport (like Http); reliability mechanisms are implemented on the application/SOAP messaging layer.

2. A Reliable transport like HTTPR is used for SOAP messaging

3. A Reliable, proprietary middleware system like IBM Websphere MQ is used for SOAP messaging.

4. A Reliable messaging standard like JMS is used for SOAP messaging. A JMS implementation is required

5. A Reliable proprietary middleware system like IBM Websphere MQ is directly used independent of SOAP
2.5.4 Assessment

Middleware endpoint to endpoint reliability

The middleware endpoint mediation essentially refers to the idea that messages are stored locally on the sender and receiver sides before and after they are being sent.

1. Option 1 does not provide this reliability since HTTP is not reliable. HTTP does not provide the status of the message on a connection failure. Therefore either SOAP messaging layer or application layer should provide the reliability.

2. SOAP over HTTPR provides the middleware endpoint to endpoint reliability. HTTPR persists the messages at the sender and receiver sides.

3. SOAP over MQ also provides the middleware endpoint to endpoint reliability. The middleware endpoints are message queue managers provided by the messaging middleware product. Unlike in the HTTPR case here the message delivery pattern is asynchronous.

4. SOAP over JMS requires a JMS implementation. Depending on the JMS implementation it provides the reliability.

5. This option also supports the reliability since underline MOM is reliable. Adapters must be used at the each side to send and receive XML messages at each side.

Application to Middleware reliability

Application to middleware reliability refers to the reliability features provided by the middleware's application to endpoint interface. This includes message delivery guarantees, fault tolerant invocation, the ability to atomically group messaging operations with other application actions.

1. When using SOAP over HTTP the reliability mechanisms may be implemented as part of the application. Application can transactionally coordinate with the message store to guarantee the reliability.

2. For options 2 – 4 applications can't communicate transactionally with MOM message store without using the MOM specific APIs.

3. For last option application to middleware reliability relates to the direct use of the underlying middleware's API and its reliability features.
Application to application reliability

Application to application reliability can be achieved in two ways.

1. In direct transaction processing, an agreement protocol is used to directly include one application's transaction processing as part of another application's transaction process. Here both applications interact with the same global transaction.

2. In Queued transaction processing two intermediate data stores can be used for sending and receiving messages. There are three transactions involved in communication between two applications. First transaction commits the message to sending data store. Then the receiving application reads the message from there and commit back to the second storage. Finally original sending application reads the response message from the second storage.
2.6 Transactions and messaging

Messaging can be integrated with the object transactions in different ways. This paper[16] pointed out such for patterns possibly used.

2.6.1 MQ Integrating Transactions

MQ Integrating transactions do the reading messages from the queue, updating the distributed object and writing the response message back to the queue in the same transaction. But this transaction corresponds only a part of the global transaction.

![Diagram of MQ Integrating Transactions]

2.6.2 Message delivery transactions

Message delivery transactions integrates the message delivery model into distributed transactions. It allows clients to send the messages asynchronously while doing the other distributed object transactions. Message delivery failures can be observed and abort the transaction accordingly. If there are messages already sent then compensation messages can be send.

Figure 2-9 MQ Integrating Transactions
2.6.3 Message processing transactions

Message processing transactions integrates the message processing model to the distributed object transactions. This enables the asynchronous request processing between transactional distributed objects. The transaction is not committed until the response is received.
2.6.4 Full messaging transactions

Full messaging transactions refer to the system which has both the message delivery transactions as well as the message processing transactions.

2.7 Middleware mediated transactions

There are two widely used transaction processing systems called. Object oriented transactions and message oriented transactions. Object oriented transactions happens in a synchronous blocking way. Further object oriented transactions uses 2PC protocol to achieve the atomicity of the transactions. In message oriented transactions only enqueuing and dequeuing messages are done transactionally. Therefore message oriented transactions does not preserve the atomicity.

Middleware mediation transactions[15] suggest a way to provide the end to end transactions while keeping the advantages of the message mediation transactions. It provides some end to end checking at the middleware layer.
2.7.1 D-sphere

D-sphere\textsuperscript{[17]} is one of the implementations of the middleware mediation transactions. D-sphere provides the end to end reliability by providing an middleware to the user which manages the end to end transactions. Following figures show how it works with and without D-sphere.

![Figure 2-12 Application Without D-sphere](image)

![Figure 2-13 Application With D-sphere](image)
D-sphere architecture supports above requirements by providing a middleware layer to users which manage transactions internally.

![D-sphere Architecture Diagram](image)

**Figure 2-14 D-sphere Architecture**