Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.1

OASIS Standard incorporating Approved Errata

12 July 2007

Specification URIs:
This Version:
http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-errata-os/wstx-wsat-1.1-spec-errata-os.html
http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-errata-os.doc
http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-errata-os.pdf

Previous Version:
http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-os/wstx-wsat-1.1-spec-os.html
http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-os.doc
http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-os.pdf

Latest Approved Version:
http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec/wstx-wsat-1.1-spec.html
http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec.doc
http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec.pdf

Technical Committee:
OASIS Web Services Transaction (WS-TX) TC

Chair(s):
Eric Newcomer, Iona
Ian Robinson, IBM

Editor(s):
Mark Little, JBoss Inc. <mark.little@jboss.com>
Andrew Wilkinson, IBM <awilkinson@uk.ibm.com>

Declared XML Namespaces:
http://docs.oasis-open.org/ws-tx/wsat/2006/06

Abstract:

The WS-AtomicTransaction specification provides the definition of the Atomic Transaction coordination type that is to be used with the extensible coordination framework described in WS-Coordination. This specification defines three specific agreement coordination protocols for the Atomic Transaction coordination type: completion, volatile two-phase commit, and durable two-phase commit. Developers can use any or all of these protocols when building applications that require consistent agreement on the outcome of short-lived distributed activities that have the all-or-nothing property.

Status:

This document was last revised or approved by the WS-TX TC on the above date. The level of approval is also listed above. Check the “Latest Approved Version” location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee’s email list. Others should send comments to the Technical Committee by using the

“Send A Comment” button on the Technical Committee’s web page at www.oasis-open.org/committees/ws-tx .

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (www.oasis-open.org/committees/ws-tx/ipr.php).

The non-normative errata page for this specification is located at www.oasis-open.org/committees/ws-tx .

Notices

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

Table of contents

1 Introduction

The current set of Web service specifications [WSDL][SOAP11][SOAP12] defines protocols for Web service interoperability. Web services increasingly tie together a number of participants forming large distributed applications. The resulting activities may have complex structure and relationships.

WS-Coordination [WSCOOR] defines an extensible framework for defining coordination types. This specification provides the definition of an Atomic Transaction coordination type used to coordinate activities having an "all or nothing" property. Atomic transactions commonly require a high level of trust between participants and are short in duration. WS-AtomicTransaction defines protocols that enable existing transaction processing systems to wrap their proprietary protocols and interoperate across different hardware and software vendors.

To understand the protocol described in this specification, the following assumptions are made:

- The reader is familiar with existing standards for two-phase commit protocols and with commercially available implementations of such protocols. Therefore this section includes only those details that are essential to understanding the protocols described.
- The reader is familiar with WS-Coordination [WSCOOR] which defines the framework for the Atomic Transaction coordination protocols.
- The reader is familiar with WS-Addressing [WSADDR] and WS-Policy [WSPOLICY].

Atomic transactions have an all-or-nothing property. The actions taken by a transaction participant prior to commit are only tentative; typically they are neither persistent nor made visible outside the transaction. When an application finishes working on a transaction, it requests the coordinator to determine the outcome for the transaction. The coordinator determines if there were any processing failures by asking the participants to vote. If the participants all vote that they were able to execute successfully, the coordinator commits all actions taken. If a participant votes that it needs to abort or a participant does not respond at all, the coordinator aborts all actions taken. Commit directs the participants to make the tentative actions final so they may, for example, be made persistent and be made visible outside the transaction. Abort directs the participants to make the tentative actions appear as if they never happened. Atomic transactions have proven to be extremely valuable for many applications. They provide consistent failure and recovery semantics, so the applications no longer need to deal with the mechanics of determining a mutually agreed outcome decision or to figure out how to recover from a large number of possible inconsistent states.

This specification defines protocols that govern the outcome of Atomic Transactions. It is expected that existing transaction processing systems will use WS-AtomicTransaction to wrap their proprietary mechanisms and interoperate across different vendor implementations.

1.1 Composable Architecture

By using the XML [XML], SOAP [SOAP11] [SOAP12] and WSDL [WSDL] extensibility model, SOAP-based and WSDL-based specifications are designed to work together to define a rich Web services environment. As such, WS-AtomicTransaction by itself does not define all features required for a complete solution. WS-AtomicTransaction is a building block used with other specifications of Web services (e.g., WS-Coordination [WSCOOR], WS-Security [WSSec]) and application-specific protocols that are able to accommodate a wide variety of coordination protocols related to the coordination actions of distributed applications.

1.2 Terminology

The uppercase key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC2119 [RFC2119].

This specification uses an informal syntax to describe the XML grammar of the XML fragments below:

- The syntax appears as an XML instance, but the values indicate the data types instead of values.
- Element names ending in "..." (such as <element.../> or <element...>) indicate that elements/attributes irrelevant to the context are being omitted.
- Attributed names ending in "..." (such as name=...) indicate that the values are specified below.
- Grammar in bold has not been introduced earlier in the document, or is of particular interest in an example.
- <!-- description --> is a placeholder for elements from some "other" namespace (like ##other in XSD).
- Characters are appended to elements, attributes, and <!-- descriptions --> as follows: "?" (0 or 1), "*" (0 or more), "+" (1 or more). The characters "[" and "]" are used to indicate that contained items are to be treated as a group with respect to the "?", "*", or "+" characters.
- The XML namespace prefixes (defined below) are used to indicate the namespace of the element being defined.
- Examples starting with <?xml contain enough information to conform to this specification; others examples are fragments and require additional information to be specified in order to conform.

1.3 Namespace

The XML namespace [XML-ns] URI that MUST be used by implementations of this specification is:

```
        http://docs.oasis-open.org/ws-tx/wsat/2006/06
```

This MUST also be used as the CoordinationContext type for Atomic Transactions.

1.3.1 Prefix Namespace

The following namespaces are used in this document:

Prefix	Namespace
S11	http://schemas.xmlsoap.org/soap/envelope
S12	http://www.w3.org/2003/05/soap-envelope
wscoor	http://docs.oasis-open.org/ws-tx/wscoor/2006/06
wsat	http://docs.oasis-open.org/ws-tx/wsat/2006/06
wsa	http://www.w3.org/2005/08/addressing

1.4 XSD and WSDL Files

Dereferencing the XML namespace defined in section 1.3 will produce the Resource Directory Description Language (RDDL) [RDDL] document that describes this namespace, including the XML schema [XML-Schema1] [XML-Schema2] and WSDL [WSDL] declarations associated with this specification.

SOAP bindings for the WSDL [WSDL], referenced in the RDDL [RDDL] document, MUST use "document" for the *style* attribute.

1.5 Protocol Elements

The protocol elements define various extensibility points that allow other child or attribute content. Additional children and/or attributes MAY be added at the indicated extension points but MUST NOT contradict the semantics of the parent and/or owner, respectively. If a receiver does not recognize an extension, the receiver SHOULD ignore the extension.

1.6 Normative References

[RDDL] Jonathan Borden, Tim Bray, eds. “Resource Directory Description Language (RDDL) 2.0”, http://www.openhealth.org/RDDL/20040118/rddl-20040118.html, January 2004

[RFC2119] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", http://www.ietf.org/rfc/rfc2119.txt, IETF RFC2119, March 1997

[SOAP11] W3C Note, "SOAP: Simple Object Access Protocol 1.1", http://www.w3.org/TR/2000/NOTE-SOAP-20000508, 08 May 2000

[SOAP12] W3C Recommendation, "SOAP Version 1.2 Part 1: Messaging Framework", http://www.w3.org/TR/soap12-part1, June 2003

[WSADDR] Web Services Addressing (WS-Addressing) 1.0, http://www.w3.org/2005/08/addressing, W3C Recommendation, May 2006

[WSCOOR] Web Services Coordination (WS-Coordination) 1.1, http://docs.oasis-open.org/ws-tx/wscoor/2006/06, OASIS, March 2006

[WSDL] Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[WSPOLICY] Web Services Policy 1.2 – Framework (WS-Policy), http://www.w3.org/Submission/2006/SUBM-WS-Policy-20060425/, W3C Member Submission, 25 April 2006.

[WSPOLICYATTACH] Web Services Policy 1.2 – Attachment (WS-PolicyAttachment), http://www.w3.org/Submission/2006/SUBM-WS-PolicyAttachment-20060425/, W3C Member Submission, 25 April 2006.

[WSSec] OASIS Standard 200401, "Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)", http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf, March 2004

[WSSecConv] Web Services Secure Conversation Language (WS-SecureConversation), http://schemas.xmlsoap.org/ws/2005/02/sc, OpenNetwork, Layer7, Netegrity, Microsoft, Reactivity, IBM, VeriSign, BEA Systems, Oblix, RSA Security, Ping Identity, Westbridge, Computer Associates, February 2005

[WSSecPolicy] Web Services Security Policy Language (WS-SecurityPolicy), http://schemas.xmlsoap.org/ws/2005/07/securitypolicy, Microsoft, VeriSign, IBM, RSA Security, July 2005

[WSTrust] Web Services Trust Language (WS-Trust), , http://schemas.xmlsoap.org/ws/2005/02/trust, OpenNetwork, Layer7, Netegrity, Microsoft, Reactivity, VeriSign, IBM, BEA Systems, Oblix, RSA Security, Ping Identity, Westbridge, Computer Associates, February 2005

[XML] W3C Recommendation, “Extensible Markup Language (XML) 1.0 (Fourth Edition)”, http://www.w3.org/TR/2006/REC-xml-20060816, 16 August 2006

[XML-ns] W3C Recommendation, "Namespaces in XML (Second Edition)", http://www.w3.org/TR/2006/REC-xml-names-20060816, 16 August 2006

[XML-Schema1] W3C Recommendation, " XML Schema Part 1: Structures Second Edition", http://www.w3.org/TR/2004/REC-xmlschema-1-20041028, 28 October 2004

[XML-Schema2] W3C Recommendation, " XML Schema Part 2: Datatypes Second Edition", http://www.w3.org/TR/2004/REC-xmlschema-2-20041028, 28 October 2004

2 Atomic Transaction Context

WS-AtomicTransaction builds on WS-Coordination [WSCOOR], which defines an Activation service, a Registration service, and a CoordinationContext type. Example message flows and a complete description of creating and registering for coordinated activities is found in WS-Coordination [WSCOOR].

The Atomic Transaction coordination context is a CoordinationContext type with the coordination type defined in this section. Atomic Transaction application messages that propagate a coordination context MUST use an Atomic Transaction coordination context. If these application messages use a SOAP binding, the Atomic Transaction coordination context MUST flow as a SOAP header in the message.

WS-AtomicTransaction adds the following semantics to the CreateCoordinationContext operation on the Activation service:

- If the request includes the CurrentContext element, the target coordinator is interposed as a subordinate to the coordinator stipulated inside the CurrentContext element.
- If the request does not include a CurrentContext element, the target coordinator creates a new transaction and acts as the root.

A coordination context MAY have an Expires element. This element specifies the period, measured from the point in time at which the context was first created or received, after which a transaction MAY be terminated solely due to its length of operation. From that point forward, the coordinator MAY elect to unilaterally roll back the transaction, so long as it has not made a commit decision. Similarly a 2PC participant MAY elect to abort its work in the transaction so long as it has not already decided to prepare.

The Atomic Transaction protocol is identified by the following coordination type:

```
        http://docs.oasis-open.org/ws-tx/wsat/2006/06
```

3 Atomic Transaction Protocols

This specification defines the following protocols for Atomic Transactions:

- **Completion:** The completion protocol initiates commit processing. Based on each protocol's registered participants, the coordinator begins with Volatile 2PC and then proceeds through Durable 2PC. The final result is signaled to the initiator.
- **Two-Phase Commit (2PC)**: The 2PC protocol coordinates registered participants to reach a commit or abort decision, and ensures that all participants are informed of the final result. The 2PC protocol has two variants:
  - **Volatile 2PC:** Participants managing volatile resources such as a cache register for this protocol.
  - **Durable 2PC:** Participants managing durable resources such as a database register for this protocol.

A participant MAY register for more than one of these protocols.

3.1 Preconditions

The correct operation of the protocols requires that a number of preconditions must be established prior to the processing:

1. The source SHOULD have knowledge of the destination's policies, if any, and the source SHOULD be capable of formulating messages that adhere to this policy.
2. If a secure exchange of messages is required, then the source and destination MUST have appropriate security credentials (such as transport-level security credentials or security tokens) in order to protect the messages.

3.2 Completion Protocol

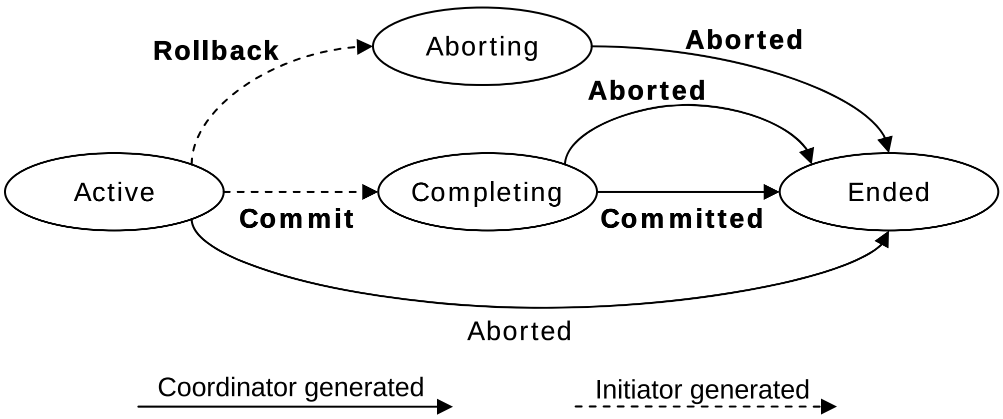
The Completion protocol is used by an application to tell the coordinator to either try to commit or abort an Atomic Transaction. After the transaction has completed, a status is returned to the application.

An initiator that registers for this protocol MUST use the following protocol identifier:

```
http://docs.oasis-open.org/ws-tx/wsat/2006/06/Completion
```

A Completion protocol coordinator MUST be the root coordinator of an Atomic Transaction. The Registration service for a subordinate coordinator MUST respond to an attempt to register for this coordination protocol with the WS-Coordination fault Cannot Register Participant.

The diagram below illustrates the protocol abstractly. Refer to section 9 State Tables for a detailed description of this protocol.



The coordinator accepts:

Commit

Upon receipt of this notification, the coordinator knows that the initiator has completed application processing. A coordinator that is Active SHOULD attempt to commit the transaction.

Rollback

Upon receipt of this notification, the coordinator knows that the initiator has terminated application processing. A coordinator that is Active MUST abort the transaction.

The initiator accepts:

Committed

Upon receipt of this notification, the initiator knows that the coordinator reached a decision to commit.

Aborted

Upon receipt of this notification, the initiator knows that the coordinator reached a decision to abort.

A coordination service that supports an Activation service MUST support the Completion protocol.

3.3 Two-Phase Commit Protocol

The Two-Phase Commit (2PC) protocol is a Coordination protocol that defines how multiple participants reach agreement on the outcome of an Atomic Transaction. The 2PC protocol has two variants: Volatile 2PC and Durable 2PC.

3.3.1 Volatile Two-Phase Commit Protocol

Upon receiving a Commit notification in the Completion protocol, the root coordinator begins the prepare phase of all participants registered for the Volatile 2PC protocol. All participants registered for this protocol MUST respond before a Prepare is issued to a participant registered for Durable 2PC. Further participants MAY register with the coordinator until the coordinator issues a Prepare to any durable participant. Once this has happened the Registration Service for the coordinator MUST respond to any further Register requests with a Cannot Register Participant fault message. A volatile recipient is not guaranteed to receive a notification of the transaction's outcome.

Participants that register for this protocol MUST use the following protocol identifier:

```
        http://docs.oasis-open.org/ws-tx/wsat/2006/06/Volatile2PC
```

3.3.2 Durable Two-Phase Commit Protocol

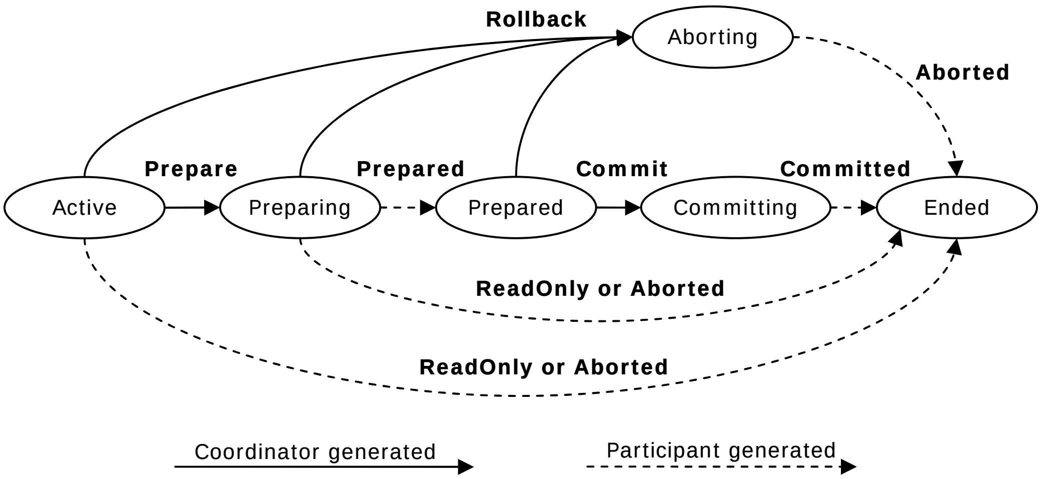
Upon successfully completing the prepare phase for Volatile 2PC participants, the root coordinator begins the prepare phase for Durable 2PC participants. All participants registered for this protocol MUST respond Prepared or ReadOnly before a Commit notification is issued to a participant registered for either protocol.

Participants that register for this protocol MUST use the following protocol identifier:

```
http://docs.oasis-open.org/ws-tx/wsat/2006/06/Durable2PC
```

3.3.3 2PC Diagram and Notifications

The diagram below illustrates the protocol abstractly. Refer to section 9 State Tables for a detailed description of this protocol.



The participant accepts:

Prepare

Upon receipt of this notification, the participant knows to enter phase one and vote on the outcome of the transaction. A participant that is Active MUST respond by sending Aborted, Prepared, or ReadOnly notification as its vote. If the participant does not know of the transaction, it MUST send an Aborted notification. If the participant knows that it has already voted, it MUST resend the same vote.

Rollback

Upon receipt of this notification, the participant knows to abort and forget the transaction. A participant that is not Committing MUST respond by sending an Aborted notification and SHOULD then forget all knowledge of this transaction. If the participant does not know of the transaction, it MUST send an Aborted notification to the coordinator.

Commit

Upon receipt of this notification, the participant knows to commit the transaction. This notification MUST only be sent after phase one and if the participant voted to commit. If the participant does not know of the transaction, it MUST send a Committed notification to the coordinator.

The coordinator accepts:

Prepared

Upon receipt of this notification, the coordinator knows the participant is Prepared and votes to commit the transaction.

ReadOnly

Upon receipt of this notification, the coordinator knows the participant votes to commit the transaction, and has forgotten the transaction. The participant does not wish to participate in phase two.

Aborted

Upon receipt of this notification, the coordinator knows the participant has aborted and forgotten the transaction.

Committed

Upon receipt of this notification, the coordinator knows the participant has committed and forgotten the transaction.

Conforming implementations MUST implement the 2PC protocol.

4 Policy Assertion

WS-Policy Framework [WSPOLICY] and WS-Policy Attachment [WSPOLICYATTACH] collectively define a framework, model and grammar for expressing the capabilities, requirements, and general characteristics of entities in an XML Web services-based system. To enable a Web service to describe transactional capabilities and requirements of a service and its operations, this specification defines an Atomic Transaction policy assertion that leverages the WS-Policy [WSPOLICY] framework.

4.1 Assertion Model

The Atomic Transaction policy assertion is provided by a Web service to qualify the transactional processing of messages associated with the particular operation to which the assertion is scoped. It indicates whether a requester MAY or MUST include an Atomic Transaction coordination context flowed with the message.

4.2 Normative Outline

The normative outline for the Atomic Transaction policy assertion is:

```
<wsat:ATAssertion [wsp:Optional="true"]? ... >
  ...
</wsat:ATAssertion>
```

The following describes additional, normative constraints on the outline listed above:

/wsat:ATAssertion

A policy assertion that specifies that an Atomic Transaction coordination context MUST be flowed inside a requester’s message. From the perspective of the requester, the target service that processes the transaction MUST behave as if it had participated in the transaction. For application messages that use a SOAP binding, the Atomic Transaction coordination context MUST flow as a SOAP header in the message.

/wsat:ATAssertion/@wsp:Optional="true"

Per WS-Policy [WSPOLICY], this is compact notation for two policy alternatives, one with and one without the assertion.

4.3 Assertion Attachment

Because the Atomic Transaction policy assertion indicates Atomic Transaction behavior for a single operation, the assertion has an Operation Policy Subject [WSPOLICYATTACH].

WS-PolicyAttachment defines two WSDL [WSDL] policy attachment points with an Operation Policy Subject:

- wsdl:portType/wsdl:operation – A policy expression containing the Atomic Transaction policy assertion MUST NOT be attached to a wsdl:portType; the Atomic Transaction policy assertion specifies a concrete behavior whereas the wsdl:portType is an abstract construct.
- wsdl:binding/wsdl:operation – A policy expression containing the Atomic Transaction policy assertion SHOULD be attached to a wsdl:binding.

4.4 Assertion Example

An example use of the Atomic Transaction policy assertion follows:

```
(01)  <wsdl:definitions
(02)      targetNamespace="bank.example.com"
```

```
(03)      xmlns:tns="bank.example.com"
(04)      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
(05)      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
(06)      xmlns:wsat="http://docs.oasis-open.org/ws-tx/wsat/2006/06"
(07)      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" >
(08)    <wsp:Policy wsu:Id="TransactedPolicy" >
(09)      <wsat:ATAssertion wsp:optional="true" />
(10)      <!-- omitted assertions -->
(11)    </wsp:Policy>
(12)    <!-- omitted elements -->
(13)    <wsdl:binding name="BankBinding" type="tns:BankPortType" >
(14)      <!-- omitted elements -->
(15)      <wsdl:operation name="TransferFunds" >
(16)        <wsp:PolicyReference URI="#TransactedPolicy" wsdl:required="true"
/>
(17)        <!-- omitted elements -->
(18)      </wsdl:operation>
(19)    </wsdl:binding>
(20)  </wsdl:definitions>

```

Lines 8-11 are a policy expression that includes an Atomic Transaction policy assertion (line 9) to indicate that an Atomic Transaction in WS-Coordination [WSCOOR] format MAY be used.

Lines 13-19 are a WSDL [WSDL] binding. Line 16 indicates that the policy in lines 8-11 applies to this binding, specifically indicating that an Atomic Transaction MAY flow inside messages.

5 Transaction Faults

Atomic Transaction faults MUST include, as the [action] property, the following fault action URI:

```
        http://docs.oasis-open.org/ws-tx/wsat/2006/06/fault
```

The protocol faults defined in this section are generated if the condition stated in the preamble is met. These faults are targeted at a destination endpoint according to the protocol fault handling rules defined for that protocol.

The definitions of faults in this section use the following properties:

[Code] The fault code.

[Subcode] The fault subcode.

[Reason] A human readable explanation of the fault.

[Detail] The detail element. If absent, no detail element is defined for the fault.

For SOAP 1.2, the [Code] property MUST be either "Sender" or "Receiver". These properties are serialized into text XML as follows:

SOAP Version	Sender	Receiver
SOAP 1.2	S12:Sender	S12:Receiver

The properties above bind to a SOAP 1.2 fault as follows:

```
<S12:Envelope>
 <S12:Header>
   <wsa:Action>
      http://docs.oasis-open.org/ws-tx/wsat/2006/06/fault
   </wsa:Action>
   <!-- Headers elided for clarity.  -->
 </S12:Header>
 <S12:Body>
  <S12:Fault>
   <S12:Code>
     <S12:Value>[Code]</S12:Value>
     <S12:Subcode>
      <S12:Value>[Subcode]</S12:Value>
     </S12:Subcode>
   </S12:Code>
   <S12:Reason>
     <S12:Text xml:lang="en">[Reason]</S12:Text>
   </S12:Reason>
   <S12:Detail>
     [Detail]
     ...
   </S12:Detail>
  </S12:Fault>
 </S12:Body>
</S12:Envelope>
```

The properties bind to a SOAP 1.1 fault as follows:

```
<S11:Envelope>
 <S11:Body>
  <S11:Fault>
   <faultcode>[Subcode]</faultcode>
   <faultstring xml:lang="en">[Reason]</faultstring>
  </S11:Fault>
 </S11:Body>
</S11:Envelope>
```

5.1 Inconsistent Internal State

This fault is sent by a participant or coordinator to indicate that a protocol violation has been detected after it is no longer possible to change the outcome of the transaction. This is indicative of a global consistency failure and is an unrecoverable condition.

Properties:

[Code] Sender

[Subcode] wsat:InconsistentInternalState

[Reason] A global consistency failure has occurred. This is an unrecoverable condition.

[Detail] Unspecified

5.2 Unknown Transaction

This fault is sent by a coordinator to indicate that it has no knowledge of the transaction and consequently cannot convey the outcome.

Properties:

[Code] Sender

[Subcode] wsat:UnknownTransaction

[Reason] The coordinator has no knowledge of the transaction. This is an unrecoverable condition.

[Detail] Unspecified

6 Security Model

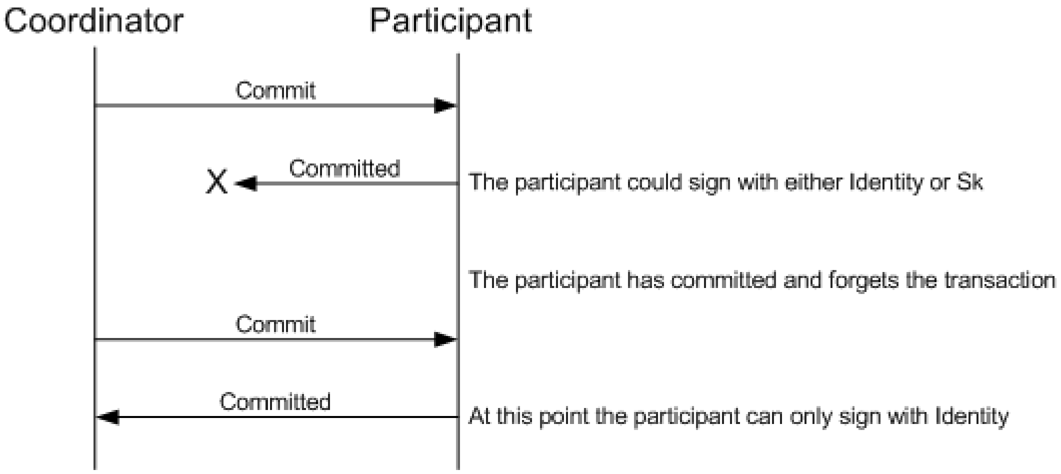
The security model for Atomic Transactions builds on the model defined in WS-Coordination [WSCOOR]. That is, services have policies specifying their requirements and requestors provide claims (either implicit or explicit) and the requisite proof of those claims. Coordination context creation establishes a base secret which can be delegated by the creator as appropriate.

Because Atomic Transactions represent a specific use case rather than the general nature of coordination contexts, additional aspects of the security model can be specified.

All access to Atomic Transaction protocol instances is on the basis of identity. The nature of transactions, specifically the uncertainty of systems means that the security context established to register for the protocol instance may not be available for the entire duration of the protocol.

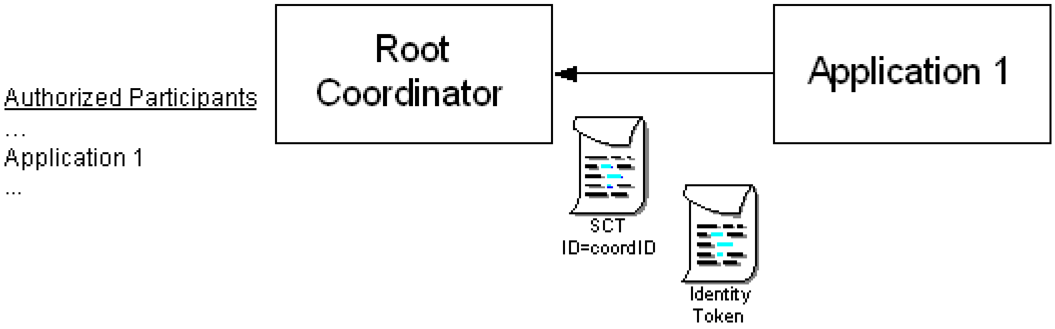
Consider, for example, the scenarios where a participant has committed its part of the transaction, but for some reason the coordinator never receives acknowledgement of the commit. The result is that when communication is re-established in the future, the coordinator will attempt to confirm the commit status of the participant, but the participant, having committed the transaction and forgotten all information associated with it, no longer has access to the special keys associated with the token.

The participant can only prove its identity to the coordinator when it indicates that the specified transaction is not in its log and assumed committed. This is illustrated in the figure below:


Coordinator
Participant
Commit
X
Committed
The participant could sign with either Identity or Sk
The participant has committed and forgets the transaction
Commit
Committed
At this point the participant can only sign with Identity

There are, of course, techniques to mitigate this situation but such options will not always be successful. Consequently, when dealing with Atomic Transactions, it is critical that identity claims always be proven to ensure that correct access control is maintained by coordinators.

There is still value in coordination context-specific tokens because they offer a bootstrap mechanism so that all participants need not be pre-authorized. As well, it provides additional security because only those instances of an identity with access to the token will be able to securely interact with the coordinator (limiting privileges strategy). This is illustrated in the figure below:



The "list" of authorized participants ensures that application messages having a coordination context are properly authorized since altering the coordination context ID will not provide additional access unless (1) the bootstrap key is provided, or (2) the requestor is on the authorized participant "list" of identities.

7 Security Considerations

It is strongly RECOMMENDED that the communication between services be secured using the mechanisms described in WS-Security [WSSec]. In order to properly secure messages, the body and all relevant headers need to be included in the signature. Specifically, the `<wscoor:CoordinationContext>` header needs to be signed with the body and other key message headers in order to "bind" the two together.

In the event that a participant communicates frequently with a coordinator, it is RECOMMENDED that a security context be established using the mechanisms described in WS-Trust [WSTrust] and WS-SecureConversation [WSSecConv] allowing for potentially more efficient means of authentication.

It is common for communication with coordinators to exchange multiple messages. As a result, the usage profile is such that it is susceptible to key attacks. For this reason it is strongly RECOMMENDED that the keys be changed frequently. This "re-keying" can be effected a number of ways. The following list outlines four common techniques:

- Attaching a nonce to each message and using it in a derived key function with the shared secret
- Using a derived key sequence and switch "generations"
- Closing and re-establishing a security context (not possible for delegated keys)
- Exchanging new secrets between the parties (not possible for delegated keys)

It should be noted that the mechanisms listed above are independent of the Security Context Token (SCT) and secret returned when the coordination context is created. That is, the keys used to secure the channel may be independent of the key used to prove the right to register with the activity.

The security context MAY be re-established using the mechanisms described in WS-Trust [WSTrust] and WS-SecureConversation [WSSecConv]. Similarly, secrets MAY be exchanged using the mechanisms described in WS-Trust [WSTrust]. Note, however, that the current shared secret SHOULD NOT be used to encrypt the new shared secret. Derived keys, the preferred solution from this list, MAY be specified using the mechanisms described in WS-SecureConversation [WSSecConv].

The following list summarizes common classes of attacks that apply to this protocol and identifies the mechanism to prevent/mitigate the attacks:

- **Message alteration** – Alteration is prevented by including signatures of the message information using WS-Security [WSSec].
- **Message disclosure** – Confidentiality is preserved by encrypting sensitive data using WS-Security [WSSec].
- **Key integrity** – Key integrity is maintained by using the strongest algorithms possible (by comparing secured policies – see WS-Policy [WSPOLICY] and WS-SecurityPolicy [WSSecPolicy]).
- **Authentication** – Authentication is established using the mechanisms described in WS-Security and WS-Trust [WSTrust]. Each message is authenticated using the mechanisms described in WS-Security [WSSec].
- **Accountability** – Accountability is a function of the type of and string of the key and algorithms being used. In many cases, a strong symmetric key provides sufficient accountability. However, in some environments, strong PKI signatures are required.
- **Availability** – Many services are subject to a variety of availability attacks. Replay is a common attack and it is RECOMMENDED that this be addressed as described in the next bullet. Other attacks, such as network-level denial of service attacks are harder to avoid and are outside the scope of this specification. That said, care should be taken to ensure that minimal processing be performed prior to any authenticating sequences.

- **Replay** – Messages may be replayed for a variety of reasons. To detect and eliminate this attack, mechanisms should be used to identify replayed messages such as the timestamp/nonce outlined in WS-Security [WSSec]. Alternatively, and optionally, other technologies, such as sequencing, can also be used to prevent replay of application messages.

8 Use of WS-Addressing Headers

The protocols defined in WS-AtomicTransaction use a "one way" message exchange pattern consisting of a sequence of notification messages between a Coordinator and a Participant. There are two types of notification messages used in these protocols:

- A notification message is a terminal message when it indicates the end of a coordinator/participant relationship. **Committed**, **Aborted** and **ReadOnly** are terminal messages, as are the protocol faults defined in this specification and in WS-Coordination [WSCOOR].
- A notification message is a non-terminal message when it does not indicate the end of a coordinator/participant relationship. **Commit**, **Rollback**, **Prepare** and **Prepared** are non-terminal messages.

The following statements define addressing interoperability requirements for the Atomic Transaction message types:

Non-terminal notification messages:

- MUST include a [source endpoint] property whose [address] property is not set to ‘http://www.w3.org/2005/08/addressing/anonymous’ or 'http://www.w3.org/2005/08/addressing/none’.

Both terminal and non-terminal notification messages:

- MUST include a [reply endpoint] property whose [address] property is set to 'http://www.w3.org/2005/08/addressing/none’.

Notification messages used in WS-AtomicTransaction protocols MUST include as the [action] property an action URI that consists of the wsat namespace URI concatenated with the "/" character and the element name of the message. For example:

```
        http://docs.oasis-open.org/ws-tx/wsat/2006/06/Commit
```

Notification messages are normally addressed according to section 3.3 of WS-Addressing 1.0 – Core [WSADDR] by both coordinators and participants using the Endpoint References initially obtained during the Register-RegisterResponse exchange. If a [source endpoint] property is present in a notification message, it MAY be used by the recipient. Cases exist where a Coordinator or Participant has forgotten a transaction that is completed and needs to respond to a resent protocol message. In such cases, the [source endpoint] property SHOULD be used as described in section 3.3 of WS-Addressing 1.0 – Core [WSADDR]. Permanent loss of connectivity between a coordinator and a participant in an in-doubt state can result in data corruption.

Protocol faults raised by a Coordinator or Participant during the processing of a notification message are terminal notifications and MUST be composed using the same mechanisms as other terminal notification messages.

All messages are delivered using connections initiated by the sender.

9 State Tables

The following state tables specify the behavior of coordinators and participants when presented with protocol messages or internal events.

Each cell in the tables uses the following convention:

Legend
Action to take Next state

Each state supports a number of possible events. Expected events are processed by taking the prescribed action and transitioning to the next state. Unexpected protocol messages MUST result in a fault message as defined in the state tables. These faults use standard fault codes as defined in either WS-Coordination [WSCOOR] or in section 5 Transaction Faults. Events that may not occur in a given state are labeled as N/A.

Notes:

1. Transitions with a “N/A” as their action are inexpressible. A TM should view these transitions as serious internal consistency issues that are likely fatal conditions.
2. The "Internal events" shown are those events, created either within a TM itself or on its local system, that cause state changes and/or trigger the sending of a protocol message.

9.1 Completion Protocol

Completion Protocol (Coordinator View)			
Inbound Events	**States**		
	None	**Active**	**Completing**
Commit	*Unknown Transaction* None	*Initiate user commit* Completing	*Ignore* Completing
Rollback	*Unknown Transaction* None	*Initiate user rollback, send aborted* None	*Invalid State* Completing
Internal Events			
Commit Decision	N/A	N/A	*Send committed* None
Abort Decision	N/A	*Send aborted* None	*Send aborted* None

9.2 2PC Protocol

These tables present the view of a coordinator or participant with respect to a single partner. A coordinator with multiple participants can be understood as a collection of independent coordinator state machines, each with its own state.

Atomic Transaction 2PC Protocol (Coordinator View)							
Inbound Events	**States**						
	None	**Active**	**Preparing**	**Prepared**	**PreparedSuccess**	**Committing**	**Aborting**
Prepared	*Durable: Send Rollback* *Volatile: Unknown Transaction* None	*Invalid State* Aborting	*Record Vote* Prepared	*Ignore* Prepared	*Ignore* PreparedSuccess	*Resend Commit* Committing	*Resend Rollback* Aborting
ReadOnly	*Ignore* None	*Forget* None	*Forget* None	*Inconsistent Internal State* Prepared	*Inconsistent Internal State* PreparedSuccess	*Inconsistent Internal State* Committing	*Forget* None
Aborted	*Ignore* None	*Forget* None	*Forget* None	*Inconsistent Internal State* Prepared	*Inconsistent Internal State* PreparedSuccess	*Inconsistent Internal State* Committing	*Forget* None
Committed	*Ignore* None	*Invalid State* Aborting	*Invalid State* Aborting	*Inconsistent Internal State* Prepared	*Inconsistent Internal State* PreparedSuccess	*Forget* None	*Inconsistent Internal State* Aborting
Internal Events							
User Commit	*N/A*	*Send Prepare* Preparing	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
User Rollback	*N/A*	*Send Rollback* Aborting	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
Expires Times Out	*N/A*	*Send Rollback* Aborting	*Send Rollback* Aborting	*Send Rollback* Aborting	*Ignore* PreparedSuccess	*Ignore* Committing	*Ignore* Aborting
Comms Times Out	*N/A*	*N/A*	*Resend Prepare* Preparing	*N/A*	*N/A*	*Resend Commit* Committing	*N/A*
Commit Decision	*N/A*	*N/A*	*N/A*	*Record Outcome* PreparedSuccess	*N/A*	*N/A*	*N/A*
Rollback Decision	*N/A*	*Send Rollback* Aborting	*Send Rollback* Aborting	*Send Rollback* Aborting	*N/A*	*N/A*	*N/A*
Write Done	*N/A*	*N/A*	*N/A*	*N/A*	*Send Commit* Committing	*N/A*	*N/A*
Write Failed	*N/A*	*N/A*	*N/A*	*N/A*	*Send Rollback* Aborting	*N/A*	*N/A*
Participant Abandoned	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	Durable: N/A Volatile: None	None

"Forget" implies that the subordinate's participation is removed from the coordinator (if necessary), and otherwise the message is ignored

Atomic Transaction 2PC Protocol (Participant View)						
Inbound Events	**States**					
	None	**Active**	**Preparing**	**Prepared**	**PreparedSuccess**	**Committing**
Prepare	*Send Aborted* None	*Gather Vote Decision* Preparing	*Ignore* Preparing	*Ignore* Prepared	*Resend Prepared* PreparedSuccess	*Ignore* Committing
Commit	*Send Committed* None	*Invalid State* None	*Invalid State* None	*Invalid State* None	*Initiate Commit Decision* Committing	*Ignore* Committing
Rollback	*Send Aborted* None	*Initiate Rollback and Send Aborted* None	*Initiate Rollback and Send Aborted* None	*Initiate Rollback and Send Aborted* None	*Initiate Rollback and Send Aborted* None	*Inconsistent Internal State* Committing
Internal Events						
Expires Times Out	*N/A*	*Initiate Rollback and Send Aborted* None	*Initiate Rollback and Send Aborted* None	*Ignore* Prepared	*Ignore* PreparedSuccess	*Ignore* Committing
Comms Times Out	*N/A*	*N/A*	*N/A*	*N/A*	*Resend Prepared* PreparedSuccess	*N/A*
Commit Decision	*N/A*	*N/A*	*Record Commit* Prepared	*N/A*	*N/A*	*Send Committed* None
Rollback Decision	*N/A*	*Send Aborted* None	*Send Aborted* None	*N/A*	*N/A*	*N/A*
Write Done	*N/A*	*N/A*	*N/A*	*Send Prepared* PreparedSuccess	*N/A*	*N/A*
Write Failed	*N/A*	*N/A*	*N/A*	*Initiate Rollback and Send Aborted* None	*N/A*	*N/A*
ReadOnly Decision	N/A	*Send ReadOnly* None	*Send ReadOnly* None	*N/A*	*N/A*	N/A

A. Acknowledgements

This document is based on initial contributions to the OASIS WS-TX Technical Committee by the following authors: Luis Felipe Cabrera (Microsoft), George Copeland (Microsoft), Max Feingold (Microsoft), Robert W Freund (Hitachi), Tom Freund (IBM), Jim Johnson (Microsoft), Sean Joyce (IONA), Chris Kaler (Microsoft), Johannes Klein (Microsoft), David Langworthy (Microsoft), Mark Little (Arjuna Technologies), Frank Leymann (IBM), Eric Newcomer (IONA), David Orchard (BEA Systems), Ian Robinson (IBM), Tony Storey (IBM), Satish Thatte (Microsoft).

The following individuals have provided invaluable input into the initial contribution: Francisco Curbera (IBM), Doug Davis (IBM), Gert Drapers (Microsoft), Don Ferguson (IBM), Kirill Gavrylyuk (Microsoft), Dan House (IBM), Oisin Hurley (IONA), Thomas Mikalsen (IBM), Jagan Peri (Microsoft), John Shewchuk (Microsoft), Stefan Tai (IBM).

The following individuals were members of the committee during the development of this specification:

Participants:

Charlton Barreto, Adobe Systems, Inc.
Martin Chapman, Oracle
Kevin Conner, JBoss Inc.
Paul Cotton, Microsoft Corporation
Doug Davis, IBM
Colleen Evans, Microsoft Corporation
Max Feingold, Microsoft Corporation
Thomas Freund, IBM
Robert Freund, Hitachi, Ltd.
Peter Furniss, Choreology Ltd.
Marc Goodner, Microsoft Corporation
Alastair Green, Choreology Ltd.
Daniel House, IBM
Ram Jeyaraman, Microsoft Corporation
Paul Knight, Nortel Networks Limited
Mark Little, JBoss Inc.
Jonathan Marsh, Microsoft Corporation
Monica Martin, Sun Microsystems
Joseph Fialli, Sun Microsystems
Eric Newcomer, IONA Technologies
Eisaku Nishiyama, Hitachi, Ltd.
Alain Regnier, Ricoh Company, Ltd.
Ian Robinson, IBM
Tom Rutt, Fujitsu Limited
Andrew Wilkinson, IBM