Web Services Business Activity (WS-Business Activity) 1.1

Public Review Draft 01, November 8, 2006

Document Identifier:
wstx-wsba-1.1-spec-pr-01

Location:
http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-pr-01.pdf

Technical Committee:
OASIS WS-TX TC

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

Editor(s):
Tom Freund, IBM <tjfreund@us.ibm.com>
Alastair Green, Choreology Ltd. <alastair.green@choreology.com>
John Harby, Independent Consultant <jharby@gmail.com>
Mark Little, JBoss Inc. <mark.little@jboss.com>

Abstract:

This specification provides the definition of the business activity coordination type that is to be used with the extensible coordination framework described in the WS-Coordination specification. The specification defines two specific agreement coordination protocols for the business activity coordination type: BusinessAgreementWithParticipantCompletion, and BusinessAgreementWithCoordinatorCompletion. Developers can use any or all of these protocols when building applications that require consistent agreement on the outcome of long-running distributed activities.

Notices

Table of Contents

1 Introduction

The current set of Web service specifications **[WSDL] [SOAP 1.1] [SOAP 1.2]** define 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.

The WS-Coordination **[WSCOOR]** specification defines an extensible framework for defining coordination types. A coordination type may have multiple coordination protocols, each intended to coordinate a different role that a Web service plays in the activity.

To establish the necessary relationships between participants, messages exchanged between participants carry a CoordinationContext. The CoordinationContext includes a Registration service Endpoint Reference of a Coordination service. Participants use that Registration service to register for one or more of the protocols supported by that activity.

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

- The reader is familiar with the WS-Coordination **[WSCOOR]** specification that defines the framework for the WS-BusinessActivity coordination protocols.
- The reader is familiar with WS-Addressing **[WSADDR]** and WS-Policy **[WSPOLICY]**.

This specification provides the definition of a business activity coordination type used to coordinate activities that apply business logic to handle exceptions that occur during the execution of activities of a business process. Actions are applied immediately and are permanent. Compensating actions may be invoked in the event of an error. The Business Activity specification defines protocols that enable existing business process and work flow systems to wrap their proprietary mechanisms and interoperate across trust boundaries and different vendor implementations.

Business Activities have the following characteristics:

- A business activity may consume many resources over a long duration.
- There may be a significant number of atomic transactions involved.
- Individual tasks within a business activity can be seen prior to the completion of the business activity, their results may have an impact outside of the computer system.
- Responding to a request may take a very long time. Human approval, assembly, manufacturing, or delivery may have to take place before a response can be sent.
- In the case where a business exception requires an Activity to be logically undone, abort is typically not sufficient. Exception handling mechanisms may require business logic, for example in the form of a compensation task, to reverse the effects of a previously completed task.
- Participants in a business activity may be in different domains of trust where all trust relationships are established explicitly.

These characteristics lead to a design point, with the following assumptions:

- All state transitions are reliably recorded, including application state and coordination metadata.
- All non-terminal notifications are acknowledged in the protocol to ensure a consistent view of state between the coordinator and participant. A coordinator or participant may solicit the status of its partner or retry sending notifications in order to achieve this.
- Each notification is defined as an individual message. Transport level request/response retry and time out are not sufficient mechanisms to achieve end-to-end agreement coordination for long-running activities.

This specification leverages WS-Coordination by extending it to support business activities. It does this by adding constraints to the protocols defined in WS-Coordination and by defining its own Coordination protocols.

The constraints that Business Activity puts on WS-Coordination protocols are described in Section 2. The Business Activity Coordination protocols are defined in Section 3.

Terms introduced in this specification are explained in the body of the specification and summarized in the Glossary.

1.1 Model

Business Activity Coordination protocols provide the following flexibility:

- A business application may be partitioned into business activity scopes. A business activity scope is a business task consisting of a general-purpose computation carried out as a bounded set of operations on a collection of Web services that require a mutually agreed outcome. There may be any number of hierarchical nesting levels. Nested scopes:
  - Allow a business application to select which child tasks are included in the overall outcome processing. For example, a business application might solicit an estimate from a number of suppliers and choose a quote or bid based on lowest-cost.
  - Allow a business application to catch an exception thrown by a child task, apply an exception handler, and continue processing even if something goes wrong. When a child completes its work, it may be associated with a compensation that is registered with the parent activity.
- A participant task within a business activity may specify that it is leaving a business activity. This provides the ability to exit a business activity and allows business programs to delegate processing to other scopes. In contrast to atomic transactions, the participant list is dynamic and a participant may exit the protocol at any time without waiting for the outcome of the protocol.
- It allows a participant task within a business activity to specify its outcome directly without waiting for solicitation. Such a feature is generally useful when a task fails so that the notification can be used by a business activity exception handler to modify the goals and drive processing in a timely manner.
- It allows participants in a coordinated business activity to perform "tentative" operations as a normal part of the activity. The result of such "tentative" operations may become visible before the activity is complete and may require business logic to run in the event that the operation needs to be compensated. Such a feature is critical when the joint work of a business activity requires many operations performed by independent services over a long period of time.

1.2 Composable Architecture

By using the XML **[XML]**,SOAP **[SOAP 1.1] [SOAP 1.2]** 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-BusinessActivity by itself does not define all features required for a complete solution. WS-BusinessActivity is a building block used with other specifications of Web services (e.g., WS-Coordination, WS-Security) and application-specific protocols that are able to accommodate a wide variety of coordination protocols related to the coordination actions of distributed applications.

1.3 Terminology

The 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][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.

XSD schemas and WSDL definitions are provided as a formal definition of grammars **[XML-Schema1] [WSDL]**.

1.4 Namespace

The XML namespace URI that MUST be used by implementations of this specification is:

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

1.4.1 Prefix Namespace

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**
wsba	**http://docs.oasis-open.org/ws-tx/wsba/2006/06**

1.5 XSD and WSDL Files

The XML schema and the WSDL declarations defined in this document can be found at the following locations:

http://docs.oasis-open.org/ws-tx/wsba/2006/06/wsba.xsd

http://docs.oasis-open.org/ws-tx/wsba/2006/06/wsba.wsdl

SOAP bindings for the WSDL documents defined in this specification MUST use "document" for the *style* attribute.

1.6 BA 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.

1.7 Normative References

[RFC2119]	S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.
[SOAP 1.1]	W3C Note, "SOAP: Simple Object Access Protocol 1.1," http://www.w3.org/TR/2000/NOTE-SOAP-20000508/, 08 May 2000.
[SOAP 1.2]	W3C Recommendation, "SOAP Version 1.2 Part 1: Messaging Framework", http://www.w3.org/TR/soap12-part1/, June 2003.

[URI]	T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 3986, http://www.ietf.org/rfc/rfc3986.txt, MIT/LCS, Day Software, Adobe Systems, January 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 1.0 (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.
[WSCOOR]	Web Services Coordination (WS-Coordination), "http:/docs.oasis-open.org/ws-tx/wscoor/2006/06"
[WSDL]	Web Services Description Language (WSDL) 1.1 "http://www.w3.org/TR/2001/NOTE-wsdl-20010315"
[WSADDR]	Web Services Addressing (WS-Addressing), Web Services Addressing (WS-Addressing) 1.0, W3C Recommendation, http://www.w3.org/2005/08/addressing
[WSPOLICY]	Web Services Policy Framework (WS-Policy), http://schemas.xmlsoap.org/ws/2004/09/policy/, VeriSign, Microsoft, Sonic Software, IBM, BEA Systems, SAP, September 2004
[WSPOLICYATTACH]	Web Services Policy Attachment (WS-PolicyAttachment), http://schemas.xmlsoap.org/ws/2004/09/policy/, VeriSign, Microsoft, Sonic Software, IBM, BEA Systems, SAP, September 2004
[BPEL]	Web Services Business Process Execution Language, http://www.oasis-open.org/committees/download.php/16024/wsbpel-specification-draft-Dec-22-2005.htm, Microsoft, BEA and IBM.
[WSSec]	OASIS Standard 200401, March 2004, "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
[WSSecPolicy]	Web Services Security Policy Language (WS-SecurityPolicy), http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/, Microsoft, VeriSign, IBM, RSA Security, December 2002
[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
[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

2 Using WS-Coordination

This section describes the Business Activity usage of WS-Coordination protocols.

2.1 Coordination Context

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

The Business Activity coordination context MUST flow on all application messages involved with the transaction.

Business Activity 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 business activity MAY be terminated solely due to its length of operation. From that point forward, the coordinator MAY elect to unilaterally cancel or compensate the activity, as appropriate, so long as it has not made a close decision. Similarly, a participant MAY elect to exit the activity so long as it has not already decided to complete.

A business activity uses the WS-Coordination CoordinationContext with the CoordinationType set to one of the following URIs:

```
http://docs.oasis-open.org/ws-tx/wsba/2006/06/AtomicOutcome
http://docs.oasis-open.org/ws-tx/wsba/2006/06/MixedOutcome
```

A CoordinationContext MAY have additional elements for extensibility.

Due to the extensibility of WS-Coordination it is also possible to define a coordination protocol type that, in addition to specifying the agreement protocol between a coordinator and a participant, also specifies the behavior of the coordination logic. For example, it may specify that the coordinator will act in an all-or-nothing manner to determine its outcome based on the outcomes communicated by its participants, or that it will use a specific majority rule when determining its final outcome based on the outcomes of its participants.

3 Coordination Types and Protocols

Business activities support two coordination types and two protocol types. Either protocol type MAY be used with either coordination type.

The coordination types are atomic and mixed as identified by the following URIs:

```
http://docs.oasis-open.org/ws-tx/wsba/2006/06/AtomicOutcome
http://docs.oasis-open.org/ws-tx/wsba/2006/06/MixedOutcome
```

A coordinator for an AtomicOutcome coordination type MUST direct all participants either to close or to compensate. A coordinator for a MixedOutcome coordination type MUST direct all participants to an outcome but MAY direct each individual participant to close or compensate. All coordinators MUST implement the AtomicOutcome coordination type. Any coordinator MAY implement the MixedOutcome coordination type.

The Coordination protocols for business activities are summarized below with names relative to the wsba base name:

- **BusinessAgreementWithParticipantCompletion**: A participant registers for this protocol with its coordinator, so that its coordinator can manage it. A participant knows when it has completed all work for a business activity.
- **BusinessAgreementWithCoordinatorCompletion**: A participant registers for this protocol with its coordinator, so that its coordinator can manage it. A participant relies on its coordinator to tell it when it has received all requests to perform work within the business activity.

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 messages.

3.2 BusinessAgreementWithParticipantCompletion Protocol

The state diagram in Figure 1 illustrates the abstract behavior of the protocol between a coordinator and a participant. The agreement coordination state reflects what each participant knows of their relationship at a given point in time. As messages take time to be delivered, the views of the coordinator and a participant may temporarily differ. Omitted are details such as resending of messages or the exchange of error messages due to protocol error.

Participants register for this protocol using the following protocol identifier:

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

The coordinator accepts:

Completed

Upon receipt of this notification, the coordinator knows that the participant has completed all processing related to the protocol instance. For the next protocol message the coordinator MUST send a Close or Compensate notification to indicate the final outcome of the protocol instance. After sending the Completed notification, a participant MUST NOT participate in any further work under that activity.

Fail

Upon receipt of this notification, the coordinator knows that the participant has failed during the Active Canceling or Compensating states; the state of the work performed by the participant is undetermined. For the next protocol message the coordinator MUST send a Failed notification. This notification carries a QName defined in schema indicating the cause of the failure.

Compensated

After transmitting this notification, the participant SHOULD forget about the activity. Upon receipt of this notification, the coordinator knows that the participant has successfully compensated all processing related to the protocol instance; the coordinator SHOULD forget its state about that participant.

Closed

After transmitting this notification, the participant SHOULD forget about the activity. Upon receipt of this notification, the coordinator knows that the participant has finalized the protocol instance successfully; the coordinator SHOULD forget its state about that participant.

Canceled

After transmitting this notification, the participant SHOULD forget about the activity. Upon receipt of this notification, the coordinator knows that the participant has successfully canceled all processing related to the protocol instance; the coordinator SHOULD forget its state about that participant.

Exit

Upon receipt of this notification, the coordinator knows that the participant will no longer participate in the business activity, and any pending work was discarded by the participant and any work performed by the participant related to the protocol instance was successfully canceled. For the next protocol message the coordinator MUST send an Exited notification. The Exit message MAY be sent by a participant only from the Active or Completing states.

CannotComplete

Upon receipt of this notification, the coordinator knows that the participant has determined that it cannot successfully complete all processing related to the protocol instance. Any pending work was discarded by the participant and any work performed by the participant related to the protocol instance was successfully canceled. For the next protocol message the coordinator MUST send a NotCompleted notification. After sending the CannotComplete notification, a participant MUST NOT participate in any further work under that activity. The CannotComplete message MAY be sent by a participant only from the Active state.

The participant accepts:

Close

Upon receipt of this notification, the participant knows the protocol instance is to be ended successfully. For the next protocol message the participant MUST send a Closed notification to end the protocol instance.

Cancel

Upon receipt of this notification, the participant knows that the work being done has to be canceled. For the next protocol message, the participant MUST send either a Canceled or Fail message. A Canceled message SHOULD be sent by the participant if the work is successfully canceled; this also ends the protocol instance. A Fail message SHOULD be sent by the participant if the work was not successfully canceled.

Compensate

Upon receipt of this notification, the participant knows that the work being done should be compensated. For the next protocol message the participant MUST send a Compensated or Fail notification to end the protocol instance. A Compensated message SHOULD be sent by the participant if the work is successfully compensated; this also ends the protocol instance. A Fail message SHOULD be sent by the participant if the work was not successfully compensated.

Failed

After transmitting this notification, the coordinator SHOULD forget about the participant. Upon receipt of this notification, the participant knows that the coordinator is aware of a failure and no further actions are required of the participant; the participant SHOULD forget the activity.

Exited

After transmitting this notification, the coordinator SHOULD forget about the participant. Upon receipt of this notification, the participant knows that the coordinator is aware the participant will no longer participate in the activity; the participant SHOULD forget the activity.

NotCompleted

After transmitting this notification, the coordinator SHOULD forget about the participant. Upon receipt of this notification, the participant knows that the coordinator is aware that the participant cannot complete all processing related to the protocol instance and that the participant will no longer participate in the activity; the participant SHOULD forget the activity.

Both the coordinator and participant accept:

GetStatus

This message requests the current state of a coordinator or participant. In response the coordinator or participant returns a Status message containing a QName indicating which column of the state table [Appendix C: State Tables for the Agreement Protocols] the coordinator or participant is currently in. GetStatus never provokes a state change.

For example, a coordinator that is waiting for a participant to initiate the BusinessAgreementWithParticipantCompletion may use this message to confirm that the participant is in one of the expected states: wsba:Active or wsba:Completed. If the participant has forgotten the activity the Status response MUST be wsba:Ended.

Status

Received in response to a getStatus request. The message includes a QName indicating the state of the Coordinator or Participant to which the request was sent. For example, if a participant is in the closing state as indicated by the state table, it would return wsba:Closing.

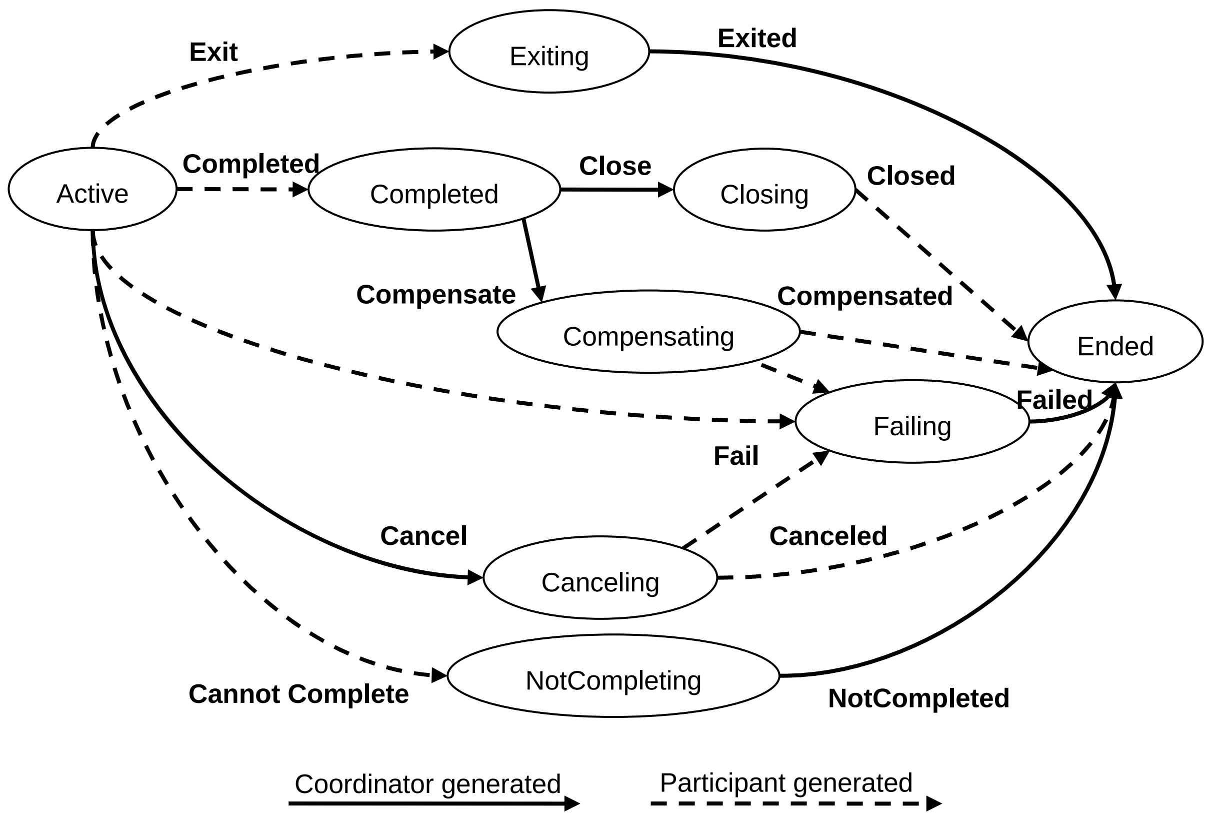

Figure 1: <u>BusinessAgreementWithParticipantCompletion abstract state diagram</u>

The coordinator may enter a condition in which it has sent a protocol message and it receives a protocol message from the participant that is consistent with the former state, not the current state. In this case, the coordinator MUST revert to the prior state, accept the notification from the participant, and continue the protocol from that point. If the participant detects this condition, it MUST discard the inconsistent protocol message from the coordinator.

A party MUST be prepared to receive duplicate notifications. If a duplicate message is received it MUST be treated as specified in the state tables described in this document.

3.3 BusinessAgreementWithCoordinatorCompletion Protocol

The BusinessAgreementWithCoordinatorCompletion protocol is the same as the BusinessAgreementWithParticipantCompletion protocol, except that a participant relies on its coordinator to tell it when it has received all requests to do work within the business activity.

Participants register for this protocol using the following protocol identifier:

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

In addition to the notifications in Section 3.1 BusinessAgreementWithParticipantCompletion Protocol above, the Business Agreement with Coordinator Completion protocol supports the following:

The coordinator accepts:

Fail

Upon receipt of this notification, the coordinator knows that the participant has failed during the Active, Canceling, Completing or Compensating states; the state of the work performed by the participant is undetermined. For the next protocol message the coordinator MUST send a Failed notification. This notification carries a QName defined in schema indicating the cause of the failure.

CannotComplete

Upon receipt of this notification, the coordinator knows that the participant has determined that it cannot successfully complete all processing related to the protocol instance. Any pending work was discarded by the participant and any work performed by the participant related to the protocol instance was successfully canceled. For the next protocol message the coordinator MUST send a NotCompleted notification. After sending the CannotComplete notification, a participant MUST NOT participate in any further work under that activity. The CannotComplete message MAY be sent by a participant only from the Active or Completing states.

The participant accepts:

Complete

Upon receipt of this notification the participant knows that it will receive no new requests for work within the business activity. The participant completes application processing and if successful MUST transmit a Completed notification. If unsuccessful the participant MUST transmit an Exit, Fail, or CannotComplete notification.

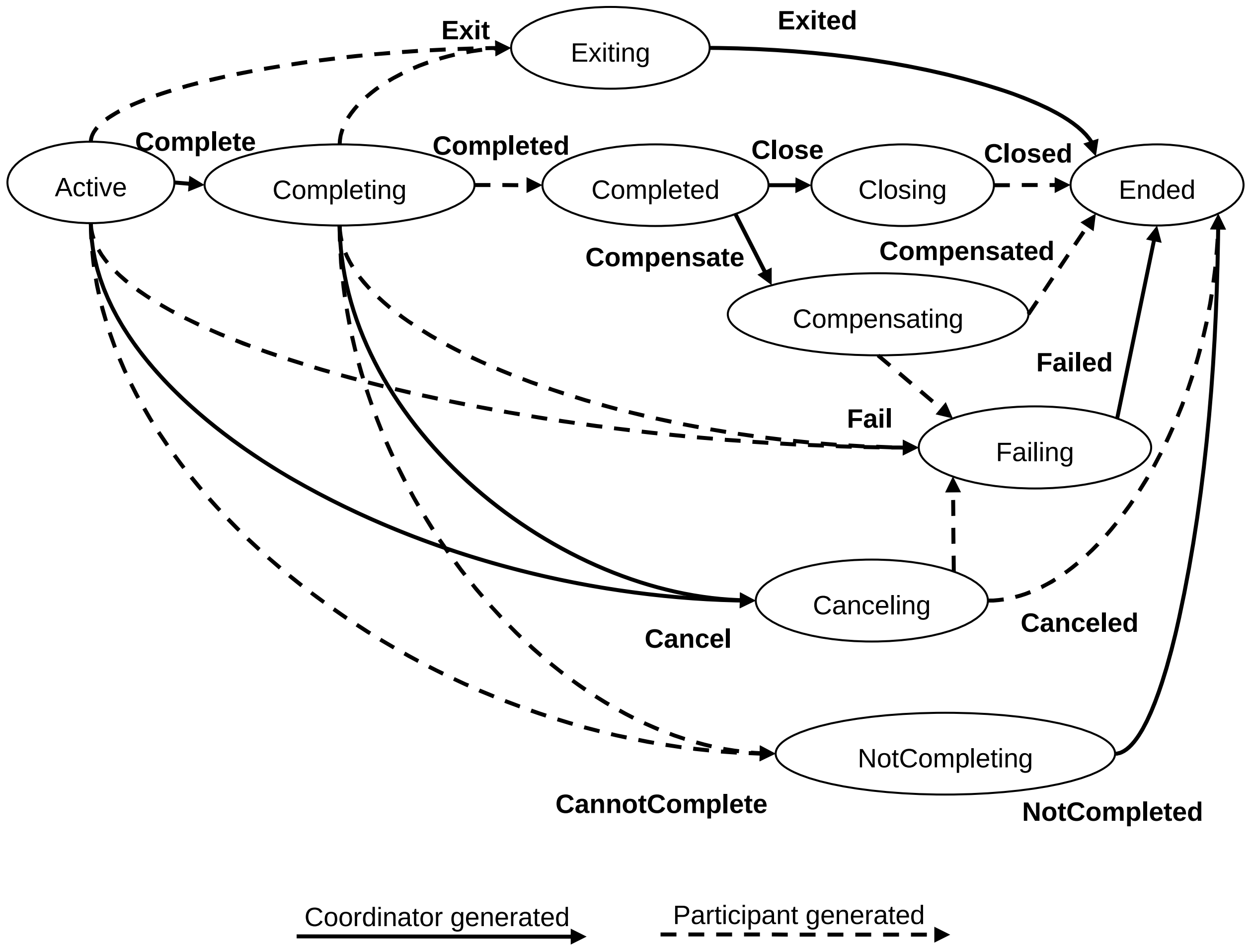

Figure 2: BusinessAgreementWithCoordinatorCompletion abstract state diagram

4 WS-BA Policy Assertions

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 business activity-related capabilities and requirements of a service and its operations, this specification defines a pair of Business Agreement policy assertions that leverage the WS-Policy framework

4.1 Assertion Models

The BA policy assertions are provided by a web service to qualify the business activity-related processing of messages associated with the particular operation to which the assertions are scoped. The BA policy assertions indicate:

whether the sender of an input message MAY or MUST include an AtomicOutcome coordination context flowed with the message. The coordination type of such a context MUST be the following:

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

whether the sender of an input message MAY or MUST include a MixedOutcome coordination context flowed with the message. The coordination type of such a context MUST be the following:

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

4.2 Normative Outlines

The normative outlines for the BA policy assertions are:

```
<wsba:BAAtomicOutcomeAssertion [wsp:Optional="true"]? ... >
  ...
</wsba:BAAtomicOutcomeAssertion>
```

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

/wsba:BAAtomicOutcomeAssertion

A policy assertion that specifies that the sender of an input message MUST include a coordination context for a business activity with AtomicOutcome coordination type flowed with the message. From the perspective of the requester, the target service that processes the transaction MUST behave as if it had participated in the transaction. The transaction MUST be represented as a SOAP header in CoordinationContext format, as defined in WS-Coordination **[WSCOOR]**.

/wsba: BAAtomicOutcomeAssertion/@wsp:Optional="true"

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

```
<wsba:BAMixedOutcomeAssertion [wsp:Optional="true"]? ... >
  ...
</wsba:BAMixedOutcomeAssertion>
```

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

/wsba:BAMixedOutcomeAssertion

A policy assertion that specifies that the sender of an input message MUST include a coordination context for a business activity with MixedOutcome coordination type flowed with the message. From the perspective of the requester, the target service that processes the transaction MUST behave as if it had participated in the transaction. The transaction MUST be represented as a SOAP header in CoordinationContext format, as defined in WS-Coordination **[WSCOOR]**.

/wsba: BAMixedOutcomeAssertion/@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 BA policy assertions indicate business activity-related behavior for a single operation, the assertions have Operation Policy Subject.

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

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

4.4 Assertion Example

An example use of the BA policy assertion follows:

```
<wsdl:definitions
  targetNamespace="hotel.example.com"
  xmlns:tns="hotel.example.com"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wsat="http://docs.oasis-open.org/ws-tx/wsba/2006/06"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
        wssecurity-utility-1.0.xsd" >
 <wsp:Policy wsu:Id="BAAtomicPolicy" >
  <wsba:BAAtomicOutcomeAssertion/>
  <!-- omitted assertions -->
 </wsp:Policy>
 <!-- omitted elements -->
 <wsdl:binding name="HotelBinding" type="tns:HotelPortType" >
  <!-- omitted elements -->
  <wsdl:operation name="ReserveRoom" >
   <wsp:PolicyReference URI="#BAAtomicPolicy"
                  wsdl:required="true" />
   <!-- omitted elements -->
  </wsdl:operation>
 </wsdl:binding>
</wsdl:definitions>
```

Lines (9-12) are a policy expression that includes a BA policy assertion (Line 10) to indicate that a coordination context for a business activity with an AtomicOutcome, expressed in WS-Coordination [WS-COOR], format MUST be used.

Lines (14-21) are a WSDL **[WSDL]** binding. Line (17) indicates that the policy in Lines (9-12) applies to this binding, specifically indicating that a coordination context for a business activity with an AtomicOutcome MUST flow inside “ReserveRoom” messages.

5 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 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 **[WSSec]** 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.

6 Use of WS-Addressing Headers

The protocols defined in WS-BusinessActivity 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. **Closed**, **Compensated**, **Canceled**, **Exited, Not Completed** and **Failed** are terminal messages as are the protocol faults defined in **[WSCOOR]**.
- A notification message is a non-terminal message when it does not indicate the end of a coordinator/participant relationship. **Complete**, **Completed**, **Close**, **Compensate**, **Cancel**, **Exit, CannotComplete** and **Fail** are non-terminal messages.

The following statements define addressing interoperability requirements for the respective WS-BusinessActivity 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-BusinessActivity MUST include as the [action] property an action URI that consists of the wsba namespace URI concatenated with the "/" character and the element name of the message. For example:

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

Notification messages are normally addressed according to section 3.3 of WS-Addressing 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.

7 Glossary

Cancel

Back out of a business activity.

Close

Terminate a business activity with a favorable outcome.

Compensate

A message to a Completed participant from a coordinator to execute its compensation. This message is part of both the BusinessAgreementWithParticipantCompletion and BusinessAgreementWithCoordinatorCompletion protocols.

Complete

A message to a participant from a coordinator telling it that it has been given all of the work for that business activity. This message is part of the BusinessAgreementWithCoordinatorCompletion protocol.

Completed

A message from a participant telling a coordinator that the participant has successfully executed everything asked of it and needs to continue participating in the protocol. This message is part of both the BusinessAgreementWithParticipantCompletion and BusinessAgreementWithCoordinatorCompletion protocols.

Exit

A message from a participant telling a coordinator that the participant does not need to continue participating in the protocol. This message is part of both the BusinessAgreementWithParticipantCompletion and BusinessAgreementWithCoordinatorCompletion protocols.

Fail

A message from a participant telling a coordinator that the participant could not execute successfully.

BusinessAgreementWithParticipantCompletion protocol

A business activity coordination protocol that supports long-lived business processes and allows business logic to handle business logic exceptions. A participant in this protocol knows when it has completed with its tasks in a business activity.

BusinessAgreementWithCoordinatorCompletion protocol

A business activity coordination protocol that supports long-lived business processes and allows business logic to handle business logic exceptions. A participant in this protocol relies on its coordinator to tell it when it has received all requests to do work within a business activity.

Scope

A business activity instance. A scope integrates coordinator and application logic. A web services application can be partitioned into a hierarchy of scopes, where the application understands the relationship between the parent scope and its child scopes.

Appendix A. Acknowledgements

This document is based on initial contribution to 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), Sean Joyce (IONA), Johannes Klein (Microsoft), David Langworthy (Microsoft), Mark Little (JBoss Inc.), 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:

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

Appendix B. Revision History

Revision	Date	Editor	Changes Made
01	11/22/2005	Tom Freund	Initial Working Draft
02	01/26/2006	Tom Freund	WS-TX: Issue #17, Specification Inconsistencies
03	03/03/2006	Tom Freund	WS-TX: Issue #7. Added resolution text WS-TX: Issue #15. Namespace & Action URI's
04	03/10/2006	Tom Freund	WS-TX: Issue #9. WS-Addressing Headers
cd-01	03/15/2006	Tom Freund	Updates to produce CD-01
05	04/xx/2006	Tom Freund	WS-TX: Action Item #31: State tables in MS-Word format WS-TX: Issue #27: Visible URL's in Reference section WS-TX: Issue #42: Swap state tables rows and columns
06	05/xx/2006	Tom Freund	WS-TX: Issue #1: Description of Status WS-TX; Issue #23: Definition of Expires WS-TX: Issue #26: Zipfile (PDF, XSD, & WSDL) WS-TX: Issue #28: Update WS-Addressing Reference WS-TX: Issue #30: One-Way Message Replies
07	06/xx/2006	Tom Freund	Accept previous changes WS-TX: Issue #45: Meaning of wsp:Optional
cd-02	06/13/2006	Tom Freund	Updates to produce CD-02
08	08/29/2006	Tom Freund	Namespace update 2006/06
09	09/14/2006	Tom Freund	WS-TX: Issue #66: Inconsistency on wsba use of fault (including wsba.wsdl & wsba.xsd) WS-TX: Issue #67: Inconsistency between schema & text (wsba.xsd) WS-TX: Issue #68: Distinguish fault conditions WS-TX: Issue #71: Remove Presumed-nothing rqmnt. WS-TX: Issue #75: Sub-Coordination undefined WS-TX: Issue #85: Protocol messages redefined WS-TX: Issue #86: Allow Fail response to Cancel msg WS-TX: Issue #87: Clarify Expires attribute WS-TX: Issue #88: Presumed-nothing contradicted WS-TX: Issue #89: Reference SOAP1.1 Addressing schema location (wsba.wsdl) Acknowledgements – Added participant list Copyright update Editorial changes

10	09/21/2006	Tom Freund	Accept all changes subsequent to CD-02
11	10/03/2006	Tom Freund	WS-TX: Issue #92: Revert to Presumed-nothing rqmnt WS-TX: Issue #94: WS-BA State Table Errata (including wsba.xsd)
cd-03	10/13/2006	Tom Freund	Updates to produce CD-03
cd-04	11/08/2006	Tom Freund	WS-TX; Issue #97: RFC 2119 Keyword updates WS-TX: Issue #98: wsdl & xsd updates WS:TX: Issue #99: Clarify wsa:Action WS:TX: Issue #102: Editoral updates WS:TX: Issue #104: Remove wsaw:Action attribute WS:TX: Issue #105: Clarify standard fault requirements

Appendix C. State Tables for the Agreement Protocols

The following state tables show state transitions that occur in the receiver when a protocol message is received or in the sender when a protocol message is sent.

Each table uses the following convention:

Each state supports a number of possible events. Expected events are processed by taking the prescribed action and transitioning of the next state. Unexpected protocol messages MUST result in a fault message as defined in the state tables. These faults MUST use a standard fault code defined in [WS-COOR].

The following rules need to be applied when reading the state tables in this document:

- For the period of time that a protocol message is in transit the sender and recipient states will be different.

  The sender of a protocol message transitions to the "next state" when the message is first sent.

  The recipient of a protocol message transitions to the "next state" when the message is first received.

- As described earlier in this document, if the coordinator receives a protocol message from the participant that is consistent with the former state of the coordinator then the coordinator reverts to its prior state, accepts the notification from the participant, and continues the protocol from that point.

The GetStatus and Status protocol messages are not included in the tables as these never result in a change of state.

C.1. Participant view of BusinessAgreementWithParticipantCompletion

BusinessAgreementWithParticipantCompletion protocol (Participant View)

Inbound Events	**States**									
	Active	**Canceling**	**Completed**	**Closing**	**Compensating**	**Failing (Active, Canceling)**	**Failing (Compensat-ing)**	**NotCompleting**	**Exiting**	**Ended**
Cancel	Canceling	*Ignore* Canceling	*Resend Completed* Completed	*Ignore* Closing	*Ignore* Compensating	*Resend Fail* Failing-*	*Ignore* Failing-Compensating	*Resend CannotComplete* NotCompleting	*Resend Exit* Exiting	*Send Canceled* Ended
Close	*Invalid State* Active	*Invalid State* Canceling	Closing	*Ignore* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* Failing-Compensating	*Invalid State* NotCompleting	*Invalid State* Exiting	*Send Closed* Ended
Compensate	*Invalid State* Active	*Invalid State* Canceling	Compensating	*Invalid State* Closing	*Ignore* Compensating	*Invalid State* Failing-*	*Resend Fail* Failing-Compensating	*Invalid State* NotCompleting	*Invalid State* Exiting	*Send Compensated* Ended
Failed	*Invalid State* Active	*Invalid State* Canceling	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	Ended	Ended	*Invalid State* NotCompleting	*Invalid State* Exiting	*Ignore* Ended
Exited	*Invalid State* Active	*Invalid State* Canceling	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* Failing-Compensating	*Invalid State* NotCompleting	Ended	*Ignore* Ended
NotCompleted	*Invalid State* Active	*Invalid State* Canceling	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* Failing-Compensating	Ended	*Invalid State* Exiting	*Ignore* Ended

BusinessAgreementWithParticipantCompletion protocol (Participant View)									
Outbound Events	**States**								
	Active	**Canceling**	**Completed**	**Closing**	**Compensating**	**Failing (Active, Canceling, Compensating)**	**NotCompleting**	**Exiting**	**Ended**
Exit	Exiting	*Invalid State* Canceling	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* NotCompleting	Exiting	*Invalid State* Ended
Completed	Completed	*Invalid State* Canceling	Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* NotCompleting	*Invalid State* Exiting	*Invalid State* Ended
Fail	Failing-Active	Failing-Canceling	*Invalid State* Completed	*Invalid State* Closing	Failing-Compensating	Failing-*	*Invalid State* NotCompleting	*Invalid State* Exiting	*Invalid State* Ended
CannotComplete	NotCompleting	*Invalid State* Canceling	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	NotCompleting	*Invalid State* Exiting	*Invalid State* Ended
Canceled	*Invalid State* Active	Ended	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* NotCompleting	*Invalid State* Exiting	Ended
Closed	*Invalid State* Active	*Invalid State* Canceling	*Invalid State* Completed	Ended	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* NotCompleting	*Invalid State* Exiting	Ended
Compensated	*Invalid State* Active	*Invalid State* Canceling	*Invalid State* Completed	*Invalid State* Closing	Ended	*Invalid State* Failing-*	*Invalid State* NotCompleting	*Invalid State* Exiting	Ended

638 ## C.2. Coordinator view of BusinessAgreementWithParticipantCompletion

639

BusinessAgreementWithParticipantCompletion protocol (Coordinator View)

Inbound Events	States: Active	Canceling	Completed	Closing	Compensating	Failing (Active, Canceling)	Failing (Compensat-ing)	NotCompleting	Exiting	Ended
Exit	Exiting	Exiting	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* Failing-Compensating	*Invalid State* NotCompleting	*Ignore* Exiting	*Resend Exited* Ended
Completed	Completed	Completed	*Ignore* Completed	*Resend Close* Closing	*Resend Compensate* Compensating	*Invalid State* Failing-*	*Ignore* Failing-Compensating	*Invalid State* NotCompleting	*Invalid State* Exiting	*Ignore* Ended
Fail	Failing-Active	Failing-Canceling	*Invalid State* Completed	*Invalid State* Closing	Failing-Compensating	*Ignore* Failing-*	*Ignore* Failing-Compensating	*Invalid State* NotCompleting	*Invalid State* Exiting	*Resend Failed* Ended
CannotComplete	NotCompleting	NotCompleting	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* Failing-Compensating	*Ignore* NotCompletng	*Invalid State* Exiting	*Resend NotCompleted* Ended
Canceled	*Invalid State* Active	Ended	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* Failing-Compensating	*Invalid State* NotCompleting	*Invalid State* Exiting	*Ignore* Ended
Closed	*Invalid State* Active	*Invalid State* Canceling	*Invalid State* Completed	Ended	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* Failing-Compensating	*Invalid State* NotCompleting	*Invalid State* Exiting	*Ignore* Ended
Compensated	*Invalid State* Active	*Invalid State* Canceling	*Invalid State* Completed	*Invalid State* Closing	Ended	*Invalid State* Failing-*	*Invalid State* Failing-Compensating	*Invalid State* NotCompleting	*Invalid State* Exiting	*Ignore* Ended

640

BusinessAgreementWithParticipantCompletion protocol
(Coordinator View)

Outbound Events	**States**								
	Active	**Canceling**	**Completed**	**Closing**	**Compensating**	**Failing (Active, Canceling, Compensating)**	**NotCompleting**	**Exiting**	**Ended**
Cancel	Canceling-Active	Canceling	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* NotCompleting	*Invalid State* Exiting	*Invalid State* Ended
Close	*Invalid State* Active	*Invalid State* Canceling	Closing	Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* NotCompleting	*Invalid State* Exiting	*Invalid State* Ended
Compensate	*Invalid State* Active	*Invalid State* Canceling	Compensating	*Invalid State* Closing	Compensating	*Invalid State* Failing-*	*Invalid State* NotCompleting	*Invalid State* Exiting	*Invalid State* Ended
Failed	*Invalid State* Active	*Invalid State* Canceling	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	Ended	*Invalid State* NotCompleting	*Invalid State* Exiting	Ended
Exited	*Invalid State* Active	*Invalid State* Canceling	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* NotCompleting	Ended	Ended
NotCompleted	*Invalid State* Active	*Invalid State* Canceling	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	Ended	*Invalid State* Exiting	Ended

C.3. Participant view of BusinessAgreementWithCoordinatorCompletion

BusinessAgreementWithCoordinatorCompletion protocol (Participant View)											
Inbound Events	**States**										
	Active	**Canceling**	**Completing**	**Completed**	**Closing**	**Compensating**	**Failing (Active, Canceling, Completing)**	**Failing (Compensat-ing)**	**NotCompleting**	**Exiting**	**Ended**
Cancel	Canceling	*Ignore* Canceling	Canceling	*Resend Completed* Completed	*Ignore* Closing	*Ignore* Compensating	*Resend Fail* Failing-*	*Ignore* Failing-Compensating	*Resend CannotComplete* NotCompleting	*Resend Exit* Exiting	*Send Canceled* Ended
Complete	Completing	*Ignore* Canceling	*Ignore* Completing	*Resend Completed* Completed	*Ignore* Closing	*Ignore* Compensating	*Resend Fail* Failing-*	*Ignore* Failing-Compensating	*Resend CannotComplete* NotCompleting	*Resend Exit* Exiting	*Send Fail* Ended
Close	*Invalid State* Active	*Invalid State* Canceling	*Invalid State* Completing	Closing	*Ignore* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* Failing-Compensating	*Invalid State* NotCompleting	*Invalid State* Exiting	*Send Closed* Ended
Compensate	*Invalid State* Active	*Invalid State* Canceling	*Invalid State* Completing	Compensating	*Invalid State* Closing	*Ignore* Compensating	*Invalid State* Failing-*	*Resend Fail* Failing-Compensating	*Invalid State* NotCompleting	*Invalid State* Exiting	*Send Compensated* Ended
Failed	*Invalid State* Active	*Invalid State* Canceling	*Invalid State* Completing	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	Ended	Ended	*Invalid State* NotCompleting	*Invalid State* Exiting	*Ignore* Ended
Exited	*Invalid State* Active	*Invalid State* Canceling	*Invalid State* Completing	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* Failing-Compensating	*Invalid State* NotCompleting	Ended	*Ignore* Ended
NotCompleted	*Invalid State* Active	*Invalid State* Canceling	*Invalid State* Completing	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* Failing-Compensating	Ended	*Invalid State* Exiting	*Ignore* Ended

BusinessAgreementWithCoordinatorCompletion protocol (Participant View)										
Outbound Events	**States**									
	Active	**Canceling**	**Completing**	**Completed**	**Closing**	**Compensating**	**Failing (Active, Canceling, Completing, Compensating)**	**NotCompleting**	**Exiting**	**Ended**
Exit	Exiting	*Invalid State* Canceling	Exiting	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* NotCompleting	Exiting	*Invalid State* Ended
Completed	*Invalid State* Active	*Invalid State* Canceling	Completed	Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* NotCompleting	*Invalid State* Exiting	*Invalid State* Ended
Fail	Failing-Active	Failing-Canceling	Failing-Completing	*Invalid State* Completed	*Invalid State* Closing	Failing-Compensating	Failing-*	*Invalid State* NotCompleting	*Invalid State* Exiting	*Invalid State* Ended
CannotComplete	NotCompleting	*Invalid State* Canceling	NotCompleting	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	NotCompleting	*Invalid State* Exiting	*Invalid State* Ended
Canceled	*Invalid State* Active	Ended	*Invalid State* Completing	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* NotCompleting	*Invalid State* Exiting	Ended
Closed	*Invalid State* Active	*Invalid State* Canceling	*Invalid State* Completing	*Invalid State* Completed	Ended	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* NotCompleting	*Invalid State* Exiting	Ended
Compensated	*Invalid State* Active	*Invalid State* Canceling	*Invalid State* Completing	*Invalid State* Completed	*Invalid State* Closing	Ended	*Invalid State* Failing-*	*Invalid State* NotCompleting	*Invalid State* Exiting	Ended

C.4. Coordinator view of BusinessAgreementWithCoordinatorCompletion

BusinessAgreementWithCoordinatorCompletion protocol
(Coordinator View)

Inbound Events	States											
	Active	**Canceling (Active)**	**Canceling (Completing)**	**Completing**	**Completed**	**Closing**	**Compensating**	**Failing (Active, Canceling, Completing)**	**Failing (Compensat-ing)**	**NotCompleting**	**Exiting**	**Ended**
Exit	Exiting	Exiting	Exiting	Exiting	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* Failing-Compensating	*Invalid State* NotCompleting	*Ignore* Exiting	*Resend Exited* Ended
Completed	*Invalid State* Active	*Invalid State* Canceling-Active	Completed	Completed	*Ignore* Completed	*Resend Close* Closing	*Resend Compensate* Compensating	*Invalid State* Failing-*	*Ignore* Failing-Compensating	*Invalid State* NotCompleting	*Invalid State* Exiting	*Ignore* Ended
Fail	Failing-Active	Failing-Canceling	Failing-Canceling	Failing-Completing	*Invalid State* Completed	*Invalid State* Closing	Failing-Compensating	*Ignore* Failing-*	*Ignore* Failing-Compensating	*Invalid State* NotCompleting	*Invalid State* Exiting	*Resend Failed* Ended
CannotComplete	NotCompleting	NotCompleting	NotCompleting	NotCompleting	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* Failing-Compensating	*Ignore* NotCompleting	*Invalid State* Exiting	*Resend NotCompleted* Ended
Canceled	*Invalid State* Active	Ended	Ended	*Invalid State* Completing	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* Failing-Compensating	*Invalid State* NotCompleting	*Invalid State* Exiting	*Ignore* Ended
Closed	*Invalid State* Active	*Invalid State* Canceling-Active	*Invalid State* Canceling-Completing	*Invalid State* Completing	*Invalid State* Completed	Ended	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* Failing-Compensating	*Invalid State* NotCompleting	*Invalid State* Exiting	*Ignore* Ended
Compensated	*Invalid State* Active	*Invalid State* Canceling-Active	*Invalid State* Canceling-Completing	*Invalid State* Completing	*Invalid State* Completed	*Invalid State* Closing	Ended	*Invalid State* Failing-*	*Invalid State* Failing-Compensating	*Invalid State* NotCompleting	*Invalid State* Exiting	*Ignore* Ended

BusinessAgreementWithCoordinatorCompletion protocol (Coordinator View)										
Outbound Events	**States**									
	Active	**Canceling (Active, Completing)**	**Completing**	**Completed**	**Closing**	**Compensating**	**Failing (Active, Canceling, Completing, Compensating)**	**NotCompleting**	**Exiting**	**Ended**
Cancel	Canceling-Active	Canceling-*	Canceling-Completing	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* NotCompleting	*Invalid State* Exiting	*Invalid State* Ended
Complete	Completing	*Invalid State* Canceling-*	Completing	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* NotCompleting	*Invalid State* Exiting	*Invalid State* Ended
Close	*Invalid State* Active	*Invalid State* Canceling-*	*Invalid State* Completing	Closing	Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* NotCompleting	*Invalid State* Exiting	*Invalid State* Ended
Compensate	*Invalid State* Active	*Invalid State* Canceling-*	*Invalid State* Completing	Compensating	*Invalid State* Closing	Compensating	*Invalid State* Failing-*	*Invalid State* NotCompleting	*Invalid State* Exiting	*Invalid State* Ended
Failed	*Invalid State* Active	*Invalid State* Canceling-*	*Invalid State* Completing	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	Ended	*Invalid State* NotCompleting	*Invalid State* Exiting	Ended
Exited	*Invalid State* Active	*Invalid State* Canceling-*	*Invalid State* Completing	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	*Invalid State* NotCompleting	Ended	Ended
NotCompleted	*Invalid State* Active	*Invalid State* Canceling-*	*Invalid State* Completing	*Invalid State* Completed	*Invalid State* Closing	*Invalid State* Compensating	*Invalid State* Failing-*	Ended	*Invalid State* Exiting	Ended