Web Services Business Activity Framework (WS-BusinessActivity)

November 2004

Authors

Luis Felipe Cabrera, Microsoft
George Copeland, Microsoft
Tom Freund, IBM
Johannes Klein, Microsoft
David Langworthy, Microsoft
Frank Leymann, IBM
David Orchard, BEA Systems
Ian Robinson, IBM
Tony Storey, IBM
Satish Thatte, Microsoft

Copyright Notice

the “Web Services Business Activity Framework” Specification will at all times remain with the Authors.

No other rights are granted by implication, estoppel or otherwise.

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.

Composable Architecture

By using the SOAP [SOAP] 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.

Status

WS-BusinessActivity and related specifications are provided for use as-is and for review and evaluation only. Microsoft, BEA, and IBM will solicit your contributions and suggestions in the near future. Microsoft, BEA, and IBM make no warrantees or representations regarding the specification in any manner whatsoever.

Acknowledgments

The following individuals have provided invaluable input into the design of the WS-Transaction specification:

Francisco Curbera, IBM
Gert Drapers, Microsoft
Doug Davis, IBM
Don Ferguson, IBM
Kirill Garvylyuk, Microsoft
Frank Leymann, IBM
Thomas Mikalsen, IBM
Jagan Peri, Microsoft
John Shewchuk, Microsoft
Stefan Tai, IBM
Sanjiva Weerawarana, IBM

We also wish to thank the technical writers and development reviewers who provided feedback to improve the readability of the specification.

Table of Contents

1 Introduction

1.1 Model

1.2 Notational Conventions
1.3 Namespace
1.4 XSD and WSDL Files
2 Using WS-Coordination
2.1 CoordinationContext
3 Coordination Types and Protocols
3.1 BusinessAgreementWithParticipantCompletion Protocol
3.2 BusinessAgreementWithCoordinatorCompletion Protocol
4 Policy
4.1. Spec Version
4.2 Protocol Types
4.3 Coordination Types
5 Security Considerations
6 Interoperability Considerations
7 Glossary
8 References
Appendix A: State Tables for the Agreement Protocols
A.1 Participant view of BusinessAgreementWithParticipantCompletion
A.2 Coodinator view of BusinessAgreementWithParticipantCompletion
A.3 Participant view of BusinessAgreementWithCoordinatorCompletion
A.4 Coordinator view of BusinessAgreementWithCoordinatorCompletion

1 Introduction

The current set of Web service specifications [WSDL] [SOAP] 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.

The WS-Coordination specification defines an extensible framework for defining coordination types. A coordination type can 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 business exceptions. 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 completed business 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 notifications are acknowledged in the protocol to ensure a consistent view of state between the coordinator and participant.
- 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 can 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 large period of time.

1.2 Notational Conventions

The keywords "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 [KEYWORDS].

Namespace URIs of the general form "some-URI" represent some application-dependent or context-dependent URI as defined in RFC2396 [URI].

1.3 Namespace

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

```
http://schemas.xmlsoap.org/ws/2004/10/wsba
```

This URI is the business coordination type identifier.

The following namespaces are used in this document:

Prefix	Namespace
S	http://www.w3.org/2003/05/soap-envelope
wscoor	http://schemas.xmlsoap.org/ws/2004/10/wscoor
wsba	http://schemas.xmlsoap.org/ws/2004/10/wsba

If an action URI is used then the action URI MUST consist of the wsba namespace URI concatenated with the "/" character and the element name. For example:

```
http://schemas.xmlsoap.org/ws/2004/10/wsba/Complete
```

1.4 XSD and WSDL Files

The following links hold the XML schema and the WSDL declarations defined in this document.

http://schemas.xmlsoap.org/ws/2004/10/wsba/wsba.xsd

http://schemas.xmlsoap.org/ws/2004/10/wsba/wsba.wsdl

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

2 Using WS-Coordination

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

2.1 CoordinationContext

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

```
http://schemas.xmlsoap.org/ws/2004/10/wsba/AtomicOutcome
http://schemas.xmlsoap.org/ws/2004/10/wsba/MixedOutcome
```

A coordination context may have an Expires attribute. This attribute specifies the earliest point in time at which a long-running activity may be terminated solely due to its length of operation. A participant could terminate its participation in the long running activity using the Exit protocol message.

A CoordinationContext can 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://schemas.xmlsoap.org/ws/2004/10/wsba/AtomicOutcome
http://schemas.xmlsoap.org/ws/2004/10/wsba/MixedOutcome
```

A coordinator for an `AtomicOutcome` coordination type must direct all participants to close or all participants to compensate. A coordinator for a `MixedOutcome` coordination type 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 must know 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 BusinessAgreementWithParticipantCompletion Protocol

The state diagram in Figure 1 specifies the 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://schemas.xmlsoap.org/ws/2004/10/wsba/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 should send a Close or Compensate notification to indicate the final outcome of the protocol instance.

Fault
: Upon receipt of this notification, the coordinator knows that the participant has failed from the active or compensating state. For the next protocol message the coordinator should send a Faulted notification. This notification carries a QName defined in schema indicating the cause of the fault.

Compensated
: Upon receipt of this notification, the coordinator knows that the participant has recorded a compensation request for a protocol.

Closed
: Upon receipt of this notification, the coordinator knows that the participant has finalized successfully.

Canceled
: Upon receipt of this notification, the coordinator knows that the participant has finalized successfully processing the Cancel notification.

Exit
: Upon receipt of this notification, the coordinator knows that the participant will no longer participate in the business activity. For the next protocol message the coordinator should send an Exited notification.

The participant accepts:

Close
: Upon receipt of this notification, the participant knows the protocol instance is to complete successfully. For the next protocol message the participant should 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 should send a Canceled notification to end the protocol instance.

Compensate
: Upon receipt of this notification, the participant knows that the work being done should be compensated. For the next protocol message the participant should send a Compensated notification to end the protocol instance.

Faulted
: Upon receipt of this notification, the participant knows that the coordinator is aware of a fault and no further actions are required of the participant.

Exited

Upon receipt of this notification, the participant knows that the coordinator is aware the participant will no longer participate in the activity.

Both the coordinator and participant accept:

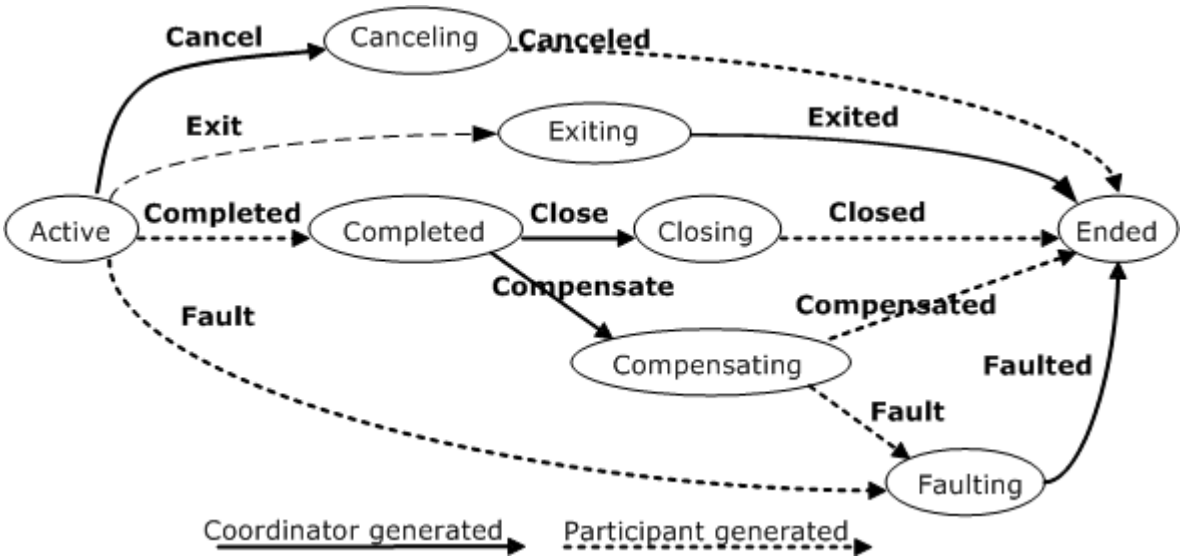
GetStatus

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 row of the state table the coordinator or participant is currently in. GetStatus never provokes a state change.

Status

Upon receipt of this message the target service returns a QName defined in schema indicating the current state of the coordinator or participant. For example, if a participant is in the closing state as indicated by the state table, it would return `wsba:Closing`.

Figure 1: BusinessAgreementWithParticipantCompletion abstract state diagram.



The coordinator can 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, it is the responsibility of the coordinator to 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 should be prepared to receive duplicate notifications. If a duplicate message is received it should be treated as specified in the state tables described in this document.

3.2 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://schemas.xmlsoap.org/ws/2004/10/wsba/CoordinatorCompletion
```

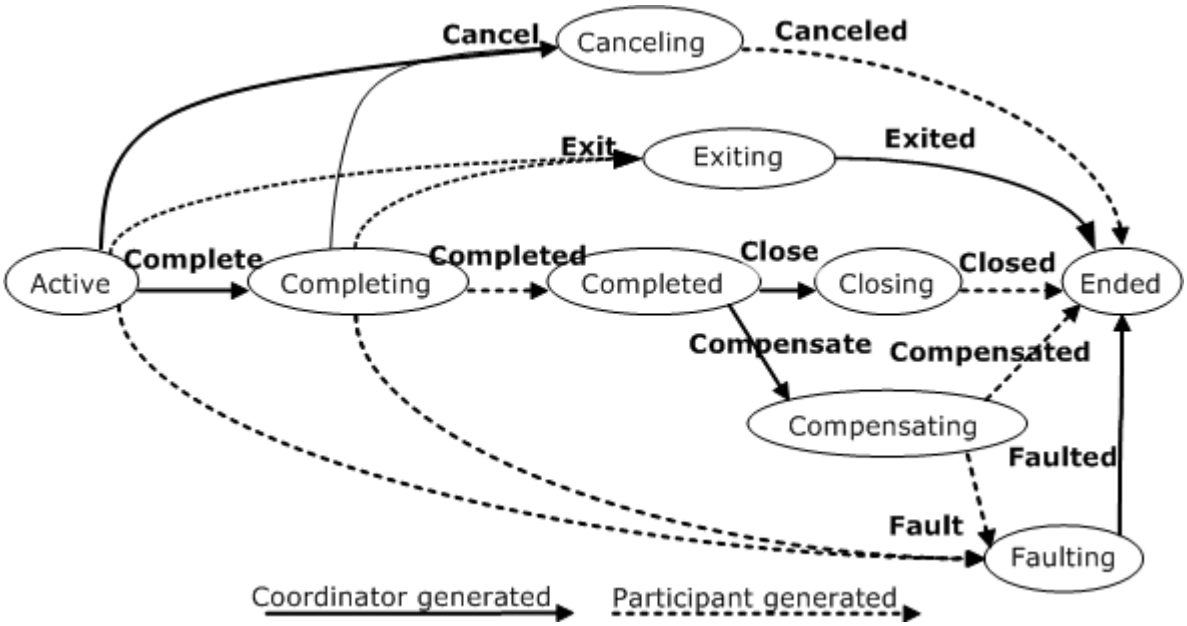
In addition to the notifications in Section 3.1, Business agreement with coordinator completion supports the following:

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. It should complete application processing and transmit the Completed notification.

Figure 2: BusinessAgreementWithCoordinatorCompletion abstract state diagram.



4 Policy

WS-Policy [WSPOLICY] defines a framework, model and grammar for expressing the capabilities, requirements, and general characteristics of entities in an XML Web services-based system. This specification leverages the WS-Policy family of specifications to enable participants and coordinators to describe and advertise their capabilities and/or requirements. The set of policy assertions for WS-Business Activity is defined below.

4.1. Spec Version

The protocol determines invariants maintained by the reliable messaging endpoints and the directives used to track and manage the delivery of messages. The assertion that will be used to identify the protocol (and version) either used or supported (depending on context) is the wsp:SpecVersion assertion that is defined in the WS-PolicyAssertions specification [WSPOLICYASSERTION].

An example use of this assertion to indicate an endpoint's support for the business activity protocol follows:

```
<wsp:SpecVersion
wsp:URI="http://schemas.xmlsoap.org/ws/2004/10/wsba"
wsp:Usage="wsp:Required"/>
```

4.2 Protocol Types

This section establishes well-known names for the protocols supported by business activities.

The following pseudo schema defines these elements:

```
<wsba:BusinessAgreementWithParticipantCompletion ... />
<wsba:BusinessAgreementWithCoordinatorCompletion ... />
```

The following describes the attributes and tags listed in the syntax above:

/wsba:BusinessAgreementWithParticipantCompletion
: This element is a policy assertion as defined in WS-PolicyAssertions. It indicates support for the protocol defined in Section 3.1

/wsba:BusinessAgreementWithParticipantCompletion
: This element is a policy assertion as defined in WS-PolicyAssertions. It indicates support for the protocol defined in Section 3.2

4.3 Coordination Types

This section establishes well-known names for the coordination types supported by business activities.

The following pseudo schema defines these elements:

```
<wsba:AtomicOutcome ... />
<wsba:MixedOutcome ... />
```

The following describes the attributes and tags listed in the syntax above:

/wsba:AtomicOutcome
: This element is a policy assertion as defined in WS-PolicyAssertions. It indicates support for the AtomicOutcome coordination type.

/wsba:MixedOutcome
: This element is a policy assertion as defined in WS-PolicyAssertions. It indicates support for the MixedOutcome coordination type.

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 can be exchanged using the mechanisms described in WS-Trust. 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, can be specified using the mechanisms described in WS-SecureConversation.

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.
- **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.

6 Interoperability Considerations

In order for two parties to communicate, both parties will need to agree on the protocols provided. This specification facilitates this agreement and thus interoperability.

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.

Fault – 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 must know 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.

8 References

[BPEL]
Web Services Business Process Execution Language, Microsoft, BEA and IBM.

[KEYWORDS]
S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, Harvard University, March 1997.

[SOAP]
W3C Note, "SOAP: Simple Object Access Protocol 1.1," 08 May 2000.

[URI]
T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998.

[XML-ns]
W3C Recommendation, "Namespaces in XML," 14 January 1999.

[XML-Schema1]
W3C Recommendation, "XML Schema Part 1: Structures," 2 May 2001.

[XML-Schema2]
W3C Recommendation, "XML Schema Part 2: Datatypes," 2 May 2001.

[WSSec]
OASIS Standard 200401, March 2004, "Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)"

[WSCOOR]
Web Services Coordination (WS-Coordination), Microsoft, IBM, and BEA Systems, October 2004

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

[WSADDR]
Web Services Addressing (WS-Addressing), Microsoft, IBM, Sun, BEA Systems, SAP, Sun, August 2004

[WSPOLICY]
Web Services Policy Framework (WS-Policy), VeriSign, Microsoft, Sonic Software, IBM, BEA Systems, SAP, September 2004

[WSPOLICYASSERTION]
Web Services Policy Assertions Language (WS-PolicyAssertions), Microsoft, IBM, BEA Systems, SAP, May 2003

Appendix A: 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:

where the next state refers to the next agreement protocol state. An Action of *Invalid State* means the sent or received protocol message cannot occur in the current state.

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-flight* 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.

A.1 Participant view of BusinessAgreementWithParticipantCompletion

BusinessAgreementWithParticipantCompletion protocol					
Participant view of state	Protocol messages received by Participant				
	Cancel	**Close**	**Compensate**	**Faulted**	**Exited**
Active	Canceling	*Invalid State* Active	*Invalid State* Active	*Invalid State* Active	*Invalid State* Active
Canceling	*Ignore* Canceling	*Invalid State* Canceling	*Invalid State* Canceling	*Invalid State* Canceling	*Invalid State* Canceling
Completed	*Resend Completed* Completed	Closing	Compensating	*Invalid State* Completed	*Invalid State* Completed
Closing	*Ignore* Closing	*Ignore* Closing	*Invalid State* Closing	*Invalid State* Closing	*Invalid State* Closing
Compensating	*Ignore* Compensating	*Invalid State* Compensating	*Ignore* Compensating	*Invalid State* Compensating	*Invalid State* Compensating
Faulting (Active, Completed)	*Resend Fault* Faulting	*Invalid State* Faulting	*Invalid State* Faulting	Ended	*Invalid State* Faulting
Faulting (Compensating)	*Ignore* Faulting	*Invalid State* Faulting	*Resend Fault* Faulting	Ended	*Invalid State* Faulting
Exiting	*Resend Exit* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	Ended
Ended	*Send Canceled* Ended	*Send Closed* Ended	*Send Compensated* Ended	*Ignore* Ended	*Ignore* Ended

BusinessAgreementWithParticipantCompletion						
Participant view of state	**Protocol messages sent by Participant**					
	Exit	**Completed**	**Fault**	**Canceled**	**Closed**	**Compensated**
Active	Exiting	Completed	Faulting-Active	*Invalid State* Active	*Invalid State* Active	*Invalid State* Active
Canceling	*Invalid State* Canceling	*Invalid State* Canceling	*Invalid State* Canceling	Ended	*Invalid State* Canceling	*Invalid State* Canceling
Completed	*Invalid State* Completed	Completed	*Invalid State* Completed	*Invalid State* Completed	*Invalid State* Completed	*Invalid State* Completed
Closing	*Invalid State* Closing	*Invalid State* Closing	*Invalid State* Closing	*Invalid State* Closing	Ended	*Invalid State* Closing
Compensating	*Invalid State* Compensating	*Invalid State* Compensating	Faulting-Compensating	*Invalid State* Compensating	*Invalid State* Compensating	Ended
Faulting	*Invalid State* Faulting	*Invalid State* Faulting	Faulting	*Invalid State* Faulting	*Invalid State* Faulting	*Invalid State* Faulting
Exiting	Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting
Ended	*Invalid State* Ended	*Invalid State* Ended	*Invalid State* Ended	Ended	Ended	Ended

A.2 Coodinator view of BusinessAgreementWithParticipantCompletion

BusinessAgreementWithParticipantCompletion						
Coordinator view of state	**Protocol messages received by Coordinator**					
	Exit	**Completed**	**Fault**	**Canceled**	**Closed**	**Compensated**
Active	Exiting	Completed	Faulting-Active	*Invalid State* Active	*Invalid State* Active	*Invalid State* Active
Canceling	Exiting	Completed	Faulting-Active	Ended	*Invalid State* Canceling	*Invalid State* Canceling
Completed	*Invalid State* Completed	*Ignore* Completed	*Invalid State* Completed	*Invalid State* Completed	*Invalid State* Completed	*Invalid State* Completed
Closing	*Invalid State* Closing	*Resend Close* Closing	*Invalid State* Closing	*Invalid State* Closing	Ended	*Invalid State* Closing
Compensating	*Invalid State* Compensating	*Resend Compensate* Compensating	Faulting-Compensating	*Invalid State* Compensating	*Invalid State* Compensating	Ended
Faulting (Compensating)	*Invalid State* Faulting	*Ignore* Faulting	*Ignore* Faulting	*Invalid State* Faulting	*Invalid State* Faulting	*Invalid State* Faulting
Faulting (Active)	*Invalid State* Faulting	*Invalid State* Faulting	*Ignore* Faulting	*Invalid State* Faulting	*Invalid State* Faulting	*Invalid State* Faulting
Exiting	*Ignore* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting
Ended	*Resend Exited* Ended	*Ignore* Ended	*Resend Faulted* Ended	*Ignore* Ended	*Ignore* Ended	*Ignore* Ended

BusinessAgreementWithParticipantCompletion protocol					
Coordinator view of state	**Protocol messages sent by Coordinator**				
	Cancel	**Close**	**Compensate**	**Faulted**	**Exited**
Active	Canceling-Active	*Invalid State* Active	*Invalid State* Active	*Invalid State* Active	*Invalid State* Active
Canceling	Canceling	*Invalid State* Canceling	*Invalid State* Canceling	*Invalid State* Canceling	*Invalid State* Canceling
Completed	*Invalid State* Completed	Closing	Compensating	*Invalid State* Completed	*Invalid State* Completed
Closing	*Invalid State* Closing	Closing	*Invalid State* Closing	*Invalid State* Closing	*Invalid State* Closing
Compensating	*Invalid State* Compensating	*Invalid State* Compensating	Compensating	*Invalid State* Compensating	*Invalid State* Compensating
Faulting	*Invalid State* Faulting	*Invalid State* Faulting	*Invalid State* Faulting	Ended	*Invalid State* Faulting
Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	Ended
Ended	*Invalid State* Ended	*Invalid State* Ended	*Invalid State* Ended	Ended	Ended

A.3 Participant view of BusinessAgreementWithCoordinatorCompletion

BusinessAgreementWithCoordinatorCompletion protocol						
Participant view of state	**Protocol messages received by Participant**					
	Cancel	**Complete**	**Close**	**Compensate**	**Faulted**	**Exited**
Active	Canceling	Completing	*Invalid State* Active	*Invalid State* Active	*Invalid State* Active	*Invalid State* Active
Canceling	*Ignore* Canceling	*Ignore* Canceling	*Invalid State* Canceling	*Invalid State* Canceling	*Invalid State* Canceling	*Invalid State* Canceling
Completing	Canceling	*Ignore* Completing	*Invalid State* Completing	*Invalid State* Completing	*Invalid State* Completing	*Invalid State* Completing
Completed	*Resend Completed* Completed	*Resend Completed* Completed	Closing	Compensating	*Invalid State* Completed	*Invalid State* Completed
Closing	*Ignore* Closing	*Ignore* Closing	*Ignore* Closing	*Invalid State* Closing	*Invalid State* Closing	*Invalid State* Closing
Compensating	*Ignore* Compensating	*Ignore* Compensating	*Invalid State* Compensating	*Ignore* Compensating	*Invalid State* Compensating	*Invalid State* Compensating
Faulting (Active, Completed)	*Resend Fault* Faulting	*Resend Fault* Faulting	*Invalid State* Faulting	*Invalid State* Faulting	Ended	*Invalid State* Faulting
Faulting (Compensating)	*Ignore* Faulting	*Ignore* Faulting	*Invalid State* Faulting	*Resend Fault* Faulting	Ended	*Invalid State* Faulting
Exiting	*Resend Exit* Exiting	*Resend Exit* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	Ended
Ended	*Send Canceled* Ended	*Ignore* Ended	*Send Closed* Ended	*Send Compensated* Ended	*Ignore* Ended	*Ignore* Ended

BusinessAgreementWithCoordinatorCompletion						
Participant view of state	**Protocol messages sent by Participant**					
	Exit	**Completed**	**Fault**	**Canceled**	**Closed**	**Compensated**
Active	Exiting	*Invalid State* Active	Faulting-Active	*Invalid State* Active	*Invalid State* Active	*Invalid State* Active
Canceling	*Invalid State* Canceling	*Invalid State* Canceling	*Invalid State* Canceling	Ended	*Invalid State* Canceling	*Invalid State* Canceling
Completing	Exiting	Completed	Faulting-Active	*Invalid State* Completing	*Invalid State* Completing	*Invalid State* Completing
Completed	*Invalid State* Completed	Completed	*Invalid State* Completed	*Invalid State* Completed	*Invalid State* Completed	*Invalid State* Completed
Closing	*Invalid State* Closing	*Invalid State* Closing	*Invalid State* Closing	*Invalid State* Closing	Ended	*Invalid State* Closing
Compensating	*Invalid State* Compensating	*Invalid State* Compensating	Faulting-Compensating	*Invalid State* Compensating	*Invalid State* Compensating	Ended
Faulting	*Invalid State* Faulting	*Invalid State* Faulting	Faulting	*Invalid State* Faulting	*Invalid State* Faulting	*Invalid State* Faulting
Exiting	Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting
Ended	*Invalid State* Ended	*Invalid State* Ended	*Invalid State* Ended	Ended	Ended	Ended

A.4 Coordinator view of BusinessAgreementWithCoordinatorCompletion

BusinessAgreementWithCoordinatorCompletion						
Coordinator view of state	**Protocol messages received by Coordinator**					
	Exit	**Completed**	**Fault**	**Canceled**	**Closed**	**Compensated**
Active	Exiting	*Invalid State* Active	Faulting-Active	*Invalid State* Active	*Invalid State* Active	*Invalid State* Active
Canceling-Active	Exiting	*Invalid State* Canceling	Faulting-Active	Ended	*Invalid State* Canceling	*Invalid State* Canceling
Canceling-Completing	Exiting	Completed	Faulting-Active	Ended	*Invalid State* Canceling	*Invalid State* Canceling
Completing	Exiting	Completed	Faulting-Active	*Invalid State* Completing	*Invalid State* Completing	*Invalid State* Completing
Completed	*Invalid State* Completed	*Ignore* Completed	*Invalid State* Completed	*Invalid State* Completed	*Invalid State* Completed	*Invalid State* Completed
Closing	*Invalid State* Closing	*Resend Close* Closing	*Invalid State* Closing	*Invalid State* Closing	Ended	*Invalid State* Closing
Compensating	*Invalid State* Compensating	*Resend Compensate* Compensating	Faulting-Compensating	*Invalid State* Compensating	*Invalid State* Compensating	Ended
Faulting (Compensating)	*Invalid State* Faulting	*Ignore* Faulting	*Ignore* Faulting	*Invalid State* Faulting	*Invalid State* Faulting	*Invalid State* Faulting
Faulting (Active, Completing)	*Invalid State* Faulting	*Invalid State* Faulting	*Ignore* Faulting	*Invalid State* Faulting	*Invalid State* Faulting	*Invalid State* Faulting
Exiting	*Ignore* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting
Ended	*Resend Exited* Ended	*Ignore* Ended	*Resend Faulted* Ended	*Ignore* Ended	*Ignore* Ended	*Ignore* Ended

BusinessAgreementWithCoordinatorCompletion protocol						
Coordinator view of state	**Protocol messages Sent by Coordinator**					
	Cancel	**Complete**	**Close**	**Compensate**	**Faulted**	**Exited**
Active	Canceling-Active	Completing	*Invalid State* Active	*Invalid State* Active	*Invalid State* Active	*Invalid State* Active
Canceling	Canceling	*Invalid State* Canceling	*Invalid State* Canceling	*Invalid State* Canceling	*Invalid State* Canceling	*Invalid State* Canceling
Completing	Canceling-Completing	Completing	*Invalid State* Completing	*Invalid State* Completing	*Invalid State* Completing	*Invalid State* Completing
Completed	*Invalid State* Completed	*Invalid State* Completed	Closing	Compensating	*Invalid State* Completed	*Invalid State* Completed
Closing	*Invalid State* Closing	*Invalid State* Closing	Closing	*Invalid State* Closing	*Invalid State* Closing	*Invalid State* Closing
Compensating	*Invalid State* Compensating	*Invalid State* Compensating	*Invalid State* Compensating	Compensating	*Invalid State* Compensating	*Invalid State* Compensating
Faulting	*Invalid State* Faulting	*Invalid State* Faulting	*Invalid State* Faulting	*Invalid State* Faulting	Ended	*Invalid State* Faulting
Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	*Invalid State* Exiting	Ended
Ended	*Invalid State* Ended	*Invalid State* Ended	*Invalid State* Ended	*Invalid State* Ended	Ended	Ended