Web Services Coordination Framework (WS-CF) Ver1.0

July 28, 2003

Authors:

Doug Bunting (doug.bunting@sun.com)
Martin Chapman (martin.chapman@oracle.com)
Oisin Hurley (ohurley@iona.com)
Mark Little (mark.little@arjuna.com) (editor)
Jeff Mischkinsky (jeff.mischkinsky@oracle.com)
Eric Newcomer (eric.newcomer@iona.com) (editor)
Jim Webber (jim.webber@arjuna.com)
Keith Swenson (KSwenson@fsw.fujitsu.com)

Abstract

Coordination is a requirement present in a variety of different aspects of distributed applications. For instance, workflow, atomic transactions, caching and replication, security, auctioning, and business-to-business activities all require some level of what may be collectively referred to as “coordination.” For example, coordination of multiple Web services in choreography may be required to ensure the correct result of a series of operations comprising a single business transaction.

Whenever coordination occurs, the propagation of additional information (the coordination context) to coordinated participants is required. The coordination context contains information such as a unique ID that allows a series of operations to share a common outcome. The outcome is typically defined in terms of coordinated state persistence operations. For example, in a Web services-based architecture, a SOAP header block might contain context information that is propagated when interacting with a coordinator, or when multiple participants exchange SOAP messages in order to create a larger interaction such as a process flow or other aggregation of services.

A Web services coordinator maintains a repository of participants and ensures that each participant receives a result of the coordinated interaction. A coordinator can also be a participant, creating a tree of sub-coordinators or peer-coordinators that cooperate to further propagate the result. When one of the participants generates a fault, for example, the coordinator ensures that all other participants are notified. A Web services coordinator sends and receives SOAP encoded messages for interoperability with any type of participant, regardless of operating system, programming language, or platform.

Context information flows as SOAP header blocks with application messages sent to participants/endpoints. The important point is that this information is specific to the type of coordination being performed, e.g., to identify the coordinator(s), the other participants, recovery information in the event of a failure, etc.

Coordination is a fundamental requirement of many distributed systems, including Web Services. However, the type of coordination protocol that is used may vary depending upon the circumstances (e.g., two-phase versus three-phase). Therefore, what is needed is a standardization on a coordination framework (coordination service) that allows users and services to register with it, and customize it on a per service or per application basis. Such a coordination service would also support newly emerging Web service standards such as workflow and transactions and builds on the Web services CTX Service.

Status of this document

This specification is a draft document and may be updated, extended or replaced by other documents if necessary. It is for review and evaluation only. The authors of this specification provide this document as is and provide no warranty about the use of this document in any case. The authors welcome feedback and contributions to be considered for updates to this document in the near future.

Table of contents

1. Introduction 7
Problem statement 7
2. WS-CF architecture 8
Extended coordination models 8
Protocol configuration and negotiation 10
Relationship to WSDL 11
3. Coordination and activities 12
Activity coordination and control 12
Coordination protocol definitions 13
4. WS-CF components 13
4.1 Participants 14
4.2 Qualifiers 18
4.3 Coordinator 19
4.3.1 Service-to-coordinator interactions 20
addParticipant 20
removeParticipant 20
getParentCoordinator 21
getQualifiers 21
4.3.2 Client-to-coordinator interactions 24
getStatus 25
Context enhancement 26
4.4 Interposition 28
State management and recovery 30
recover 31
getStatus 32
5. Roles & Responsibilities 33
Coordination Service Activity Lifecycle Service provider 34
Coordination Service Provider 34
Web Service Provider 35
6. Example 36

7. XML Schema and WSDL Interfaces 37
7.1 XML Schema for WS-CF Messages........ 38
7.2 WSDL Interface for WS-CF Actors 43
7.3 Issues........ 57
8. References........ 61
9. Acknowledgements........ 63

1. Introduction

Coordination is the act of one agent (the *coordinator*) disseminating information to a number of *participants* to guarantee that all participants obtain a specific message. A coordinator can accept the responsibility, for example, of notifying all participants in an Activity of a common outcome.

Coordination is a fundamental requirement in distributed systems that many applications use either explicitly or implicitly, e.g., workflow, atomic transactions, caching and replication, security, auctioning, and business-to-business activities. Coordination propagates additional information (the *coordination context*) to the participants.

Context information can flow implicitly (transparently to the application) within normal messages sent to the participants, or it may be an explicit action on behalf of the client/service. This information is specific to the type of coordination being performed, e.g., to identify the coordinator(s), the other participants in an Activity, recovery information in the event of a failure, etc. Furthermore, it may be required that additional application specific context information (e.g.. extra SOAP header information) flow to these participants or the services which use them.

Coordination is an integral part of any distributed system, but there is no single type of coordination protocol that can suffice for all problem domains. Therefore, what is needed is a common *Web Services Coordination Framework* (*WS-CF*) that allows users and services to tie into it and customize it on a per service or application basis. A suitably designed coordination service should provide enough flexibility and extensibility to its users that allow it to be tailored, statically or dynamically, to fit any requirement.

This service builds upon WS-CTX and supports WS-TXM, as well as other Web Service standards in the area of choreography, workflow and transactions. In the case of transactions, for example, unlike other attempts which are solutions to one specific problem area and are therefore not applicable to others, different extended transaction models can be relatively easily developed to suit specific domains, and interoperability across transaction protocols supported.

This specification presents the outline of such a service.

Problem statement

Define a specification for a generic coordination service for a Web Services, to be known as the WS-CF, utilizing the Web Services CTX Service specification for the definition of basic activities (i.e., determining the scope of shared context). Outline the necessary infrastructure and protocol requirements to support a coordination service for interacting with the participants in one or more Activities. A coordinator can also be a participant to another coordinator, extending the ability to interoperate across application domains.

Coordinators are themselves modeled as Web services and can be combined into multiple-coordinator patterns to extend and optimize the supported interaction patterns.

The WS-CF is designed to be used together with and to compliment other Web services technologies such as reliable messaging, routing, inspection, security, and process flow.

The goals of the specification are to:

- Provide a basic definition of a core infrastructure service consisting of a Coordinator Service for the Web Service environment. WS-CF that builds on the Web Services CTX Service.
- Define the mappings onto the Web Service environment (SOAP message and header definitions, context definition, endpoint address requirements, etc.).
- Define the required infrastructure support such as event mechanisms, etc.
- Define the roles and responsibilities of WS-CF subcomponents (e.g., Coordination Service Participants).

2. WS-CF architecture

The following sections outline the architecture of WS-CF, describing the components that implementations provide and those that are required from users.

Extended coordination models

The WS-CF allows the *management and coordination* in a Web services interaction of a number of *activities* related to an overall application. It builds on the Web Services CTX Service (WS-CTX) specification and provides a coordination service that plugs into WS-CTX. In particular WS-CF:

- Defines demarcation points which specify the start and end points of coordinated activities; this is done automatically by invoking an Activity;
- Defines demarcation points where coordination of participants occurs (i.e., at which points the appropriate SOAP messages are sent to participants);
- Registers participants for the activities that are associated with the application;
- Propagates coordination-specific information across the network by enhancing the default context structure provided by WS-CTX;

The main components involved in using and defining the WS-CF are:

1) A *Coordinator*: Provides an interface for the registration of participants (such as *activities*) triggered at *coordination points*. The coordinator is responsible for communicating the outcome of the activity to the list of registered activities. Importantly, coordination is not restricted to the end of an activity: an activity can execute (different) coordination protocols at arbitrary points during its lifetime.

Coordination extends the notion of an *activity* to represent a defined set of tasks with a set of related *coordination actions*;

2) A *Participant*: The operation or operations that are performed as part of coordination sequence processing

3) A *Coordination Service*: Defines the behaviour for a specific coordination model. The Coordination Service provides a processing pattern that is used for outcome processing. For example, an ACID transaction service is one implementation of a Coordination Service that provides a two-phase protocol definition whose coordination sequence processing includes *Prepare*, *Commit* and *Rollback*. Other examples of Coordination Service implementations include extended transaction patterns such as Sagas, Collaborations, Nested or Real-Time transactions and non-transactional patterns such as Cohesions and Correlations. Coordination can also be used to group related non-transactional activities. Multiple Coordination Service implementations may co-exist within the same application and processing domain. WS-CF does not specify how a Coordination Service is implemented. For example, a given implementation may support multiple coordination protocols as in [1].

As we shall show, WS-CF uses the Coordinator and Participant roles to define coordination protocols and associated message sets. However, in order to support existing coordination services which may have already defined coordinator and participant interfaces and message sets, a WS-CF compliant implementation is only required to provide an implementation of the Activity Lifecycle Service. This allows the coordinator to be tied to activities and to augment the basic WS-CTX context. It is assumed that in the absence of WS-CF Coordinator Service and Participants, the interfaces to these services and protocol message sets are defined elsewhere and known by users/services. In the remainder of this specification we shall only consider the specific case of protocols using all of the roles defined by WS-CF.

Figure 1 shows the various WS-CF services and their relationships to one another and WS-CTX. Web services are shown as circles. The mandated WS-CF services are the CoordinationServiceALS and the CTX Service, whereas the optional services which may be provided through non-WS-CF routes are the Application Web Service, Coordination Service and Participant.

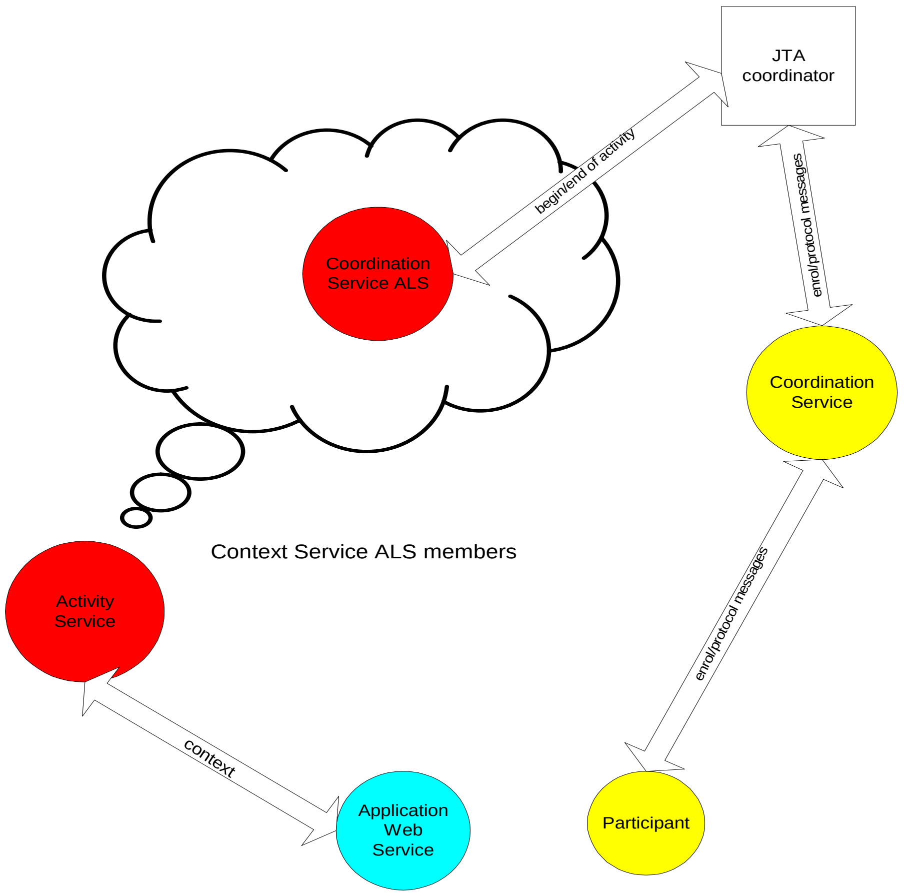

Figure 1, WS-CF services.

Protocol configuration and negotiation

It is possible that Web Service components may support multiple different Coordination Service models (possibly representing different qualities of service). Either when the Web application is created, or when one component initially interacts with another, some level of protocol negotiation will be necessary to determine which transaction model will be used. If the component does not support the required Coordination Service model then it will be up to the application to determine whether or not it makes sense to continue to use the component. For example, it may make sense for a transactional application to refuse to work with any service that does not support transactional semantics, i.e., does not accept (and use) transaction contexts that may be sent to it.

Additionally, the operational service protocol message exchange includes the requirement for a means to:

- Allow a protocol message exchange independent of normal message exchange.
- A means to perform outcome processing (an identity for direct communication between coordinator and participant(s)).

It is important that the negotiation and protocol exchange mechanisms not place any additional requirement on the transport.

Note, such requirements do not preclude the reuse of existing product implementations. However, it must be recognized that when using a common Web Service definition to communicate between operational domains that messages exchanges may need to decomposed into their constituent parts, i.e., a phase to establish and exchange service information and context and a phase for the operational message.

In addition, we do not assume that a single remote invocation mechanism (e.g., HTTP) will be the natural communication medium for all Web Services. How participants within and between activities appear to each other is not central to this discussion. They may be services communicating via HTTP with WS-CF information traveling via SMTP, for example. We assume that they will use the most appropriate invocation protocol for the application. This does not preclude a given application from using multiple object models and communication protocols simultaneously.

Relationship to WSDL

Where WSDL is used in this specification we shall use a synchronous invocation style for sending requests. In order to provide for loose-coupling of entities all responses are sent using synchronous call-backs. However, this is not prescriptive and other binding styles are possible.

For clarity WSDL is shown in an abbreviated form in the main body of the document: only *portTypes* are illustrated; a default binding to SOAP 1.1-over-HTTP is also assumed as per [2]. Complete WSDL is available at the end of the specification.

3. Coordination and activities

In the WS-CTX specification it was shown how the framework manages the lifecycle of Activities, which are used to scope application and service specific work, along with the associated Activity contexts necessary for distributed invocations. It also described how services can be plugged into this framework in order that they can enhance it at necessary stages in the lifecycle of an Activity. In this section a specific service (coordination), which is integral to the development of Web Services management, is presented. This service is more accurately described as a framework that supports arbitrary coordination protocols; the intention is that such protocols can be plugged into the framework to customize it for other application and service requirements, e.g., by adding a two-phase protocol for consensus or a three-phase protocol if operating in a particularly failure-prone or untrustworthy environment. This is also the first high-level service to be added to the core Context Service framework. It is our intention that other services can then use coordination for their own purposes, e.g., transactions.

Coordination is the act of an entity (the *coordinator*) disseminating information to a number of *participants* for a variety of reasons, e.g., in order to reach consensus on a decision, or simply to guarantee that all participants obtain a specific message. Coordination is a fundamental requirement in distributed systems that many applications use either explicitly or implicitly, e.g., workflow, atomic transactions, caching and replication, security, auctioning, and business-to-business activities. Whenever coordination occurs, the propagation of additional information (the *coordination context*) to coordinated participants is also required.

WS-CF defines the scope of an activity to be the scope of a coordinated interaction: upon termination of an activity, the associated coordinator will be contacted in order that it can execute the coordination protocol. Depending upon the coordination protocol, coordination may also occur at arbitrary points during the lifetime of an individual activity, but this need not be supported by all implementations.

Activity coordination and control

An activity may run for an arbitrary length of time and may need to use coordination at any number of points during its lifetime. For example, consider Figure 2, which shows a series of connected activities co-operating during the lifetime of an application. The darker ellipses represent coordination boundaries, whereas the lighter ellipses delimit activity boundaries. Activity *A1* uses two coordination points during its execution, whereas *A2* uses none. Additionally, coordinated activity *A3* has another coordinated activity, *A3'* nested within it. The activity service and coordination framework combination is responsible for distributing both the activity and coordination contexts between execution environments in order that the hierarchy can be fully distributed.

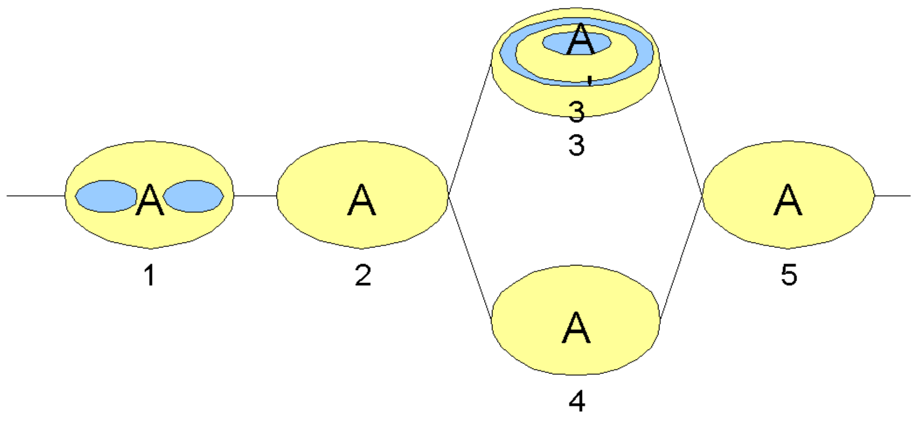

Figure 2, Activity and Transaction Relationship.

The coordinator associated with an activity is allowed to change during the lifetime of the activity, to reflect the changing requirements of activities. For example, in the diagram above, at the first coordination point A1 may use a two-phase protocol to achieve consensus, whereas when the activity terminates, a three phase protocol may be more appropriate. How activities are coordinated is the domain of the *Coordination Service*. It does this by utilizing the components described in the following sections.

Coordination protocol definitions

A coordination protocol is defined by the message interactions between the coordinator and its participants, and the semantics that are imposed on those interactions. It is beyond the scope of this specification to manage semantic information about individual protocol types. Coordination protocols are unambiguously identified by a URI. It is also beyond the scope of the specification to indicate how coordinator implementations are located or associated with their URIs.

4. WS-CF components

The components are described in terms of their *behaviour and the interactions* that occur between them. All interactions are described in terms of messages, which an implementation may abstract at a higher level into request/response pairs or RPCs, for example. As such, all communicated messages are required to contain response endpoint addresses solely for the purposes of each interaction.

One consequence of these interactions is that faults and errors which may occur when a service is invoked are communicated back to interested parties via messages which are themselves part of the protocol. For example, if an operation might fail because no activity is present when one is required, then it will be valid for the *noActivityFault* message to be received by the response service. To accommodate other errors or faults, all response service signatures have a *generalFault* operation.

Note, in the rest of this section we will use the term "invokes operation X on service Y" when referring to invoking services. This term does not imply a specific implementation for performing such service invocations and is used merely as a short-hand for "sends message X to service Y." As long as implementations ensure that the on-the-wire message formats are compliant with those defined in this specification, how the endpoints are implemented and how they expose the various operations (e.g., via WSDL [2]) is not mandated by this specification.

4.1 Participants

At coordination points defined by the application or service, messages are communicated between a coordinator and registered participants through the exchange of *protocol specific* messages. For example, the termination of one activity may initiate the start/restart of other activities in a workflow-like environment. Messages can be used to infer a flow of control during the execution of an application. The information encoded within a message will depend upon the implementation of the coordination model.

A Participant (coordination participant) will use the message in a manner specific to the Coordination Service and return a result of it having done so. For example, upon receipt of a specific message, a Participant could start another activity running (e.g., a compensation activity); another Participant could commit any modifications to a database when it receives one type of message, or undo them if it receives another type.

Each participant supports a coordination protocol specific to the model implemented by the coordinator (e.g., two-phase commit). In addition, the work that a participant performs when it receives a message from the coordinator is dependent on the participant's implementation (e.g., to commit the reservation of the theatre ticket and debit the user's account).

Interactions for executing a coordination protocol are broken down into two distinct types (these messages are all contextualized unless otherwise noted):

- Coordinator-to-participant, where the coordinator sends a protocol message to the participant and will eventually get a response.
- Participant-to-coordinator, where the participant may autonomously communicate protocol messages to the coordinator.

In order to perform the necessary interactions for coordinator-to-participant, two service roles are defined (illustrated in Figure 3), with the following operations (messages):

- The Participant: this accepts *getStatus*, *AssertionType* and *getIdentity* messages. The CoordinatorParticipant endpoint address is propagated on all of these messages.
- The CoordinatorParticipant: this accepts *status*, *AssertionType*, *identity*, *wrongState* and *generalFault* call-back messages. Other error or fault messages are expected to be returned as specific instances of the AssertionType response.

The coordinator sends an *AssertionType* message to the Participant with an accompanying reference to a CoordinatorParticipant to which the Participant may eventually call-back with the response. The Participant may then send back a specific AssertionType message if successful, which will be interpreted in a manner specific to the coordination protocol. The *wrongState* and *generalFault* messages are used to indicate error conditions.

The *getIdentity* message is used to obtain the unique identification for the relevant Participant.

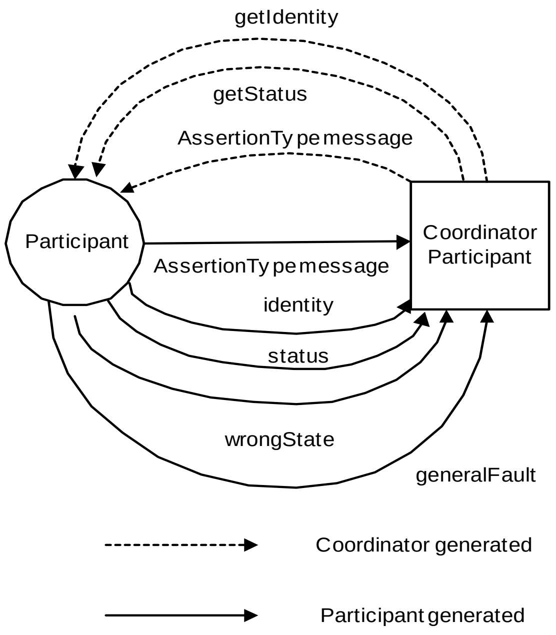

Figure 3, Coordinator-to-participant interactions.

The interactions depicted in Figure 3, are presented on a per-role basis in the WSDL interface shown in Figure 4.

```
<wsdl:portType name="ParticipantPortType">
  <wsdl:operation name="getStatus">
    <wsdl:input message="tns:GetStatusMessage"/>
  </wsdl:operation>
  <wsdl:operation name="getIdentity">
    <wsdl:input message="tns:GetIdentityMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="CoordinatorParticipantPortType">
  <wsdl:operation name="status">
    <wsdl:input message="tns:StatusMessage"/>
  </wsdl:operation>
  <wsdl:operation name="Identity">
    <wsdl:input message="tns:IdentityMessage"/>
  </wsdl:operation>
  <wsdl:operation name="wrongState">
    <wsdl:input message="asw:WrongStateFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="generalFault">
    <wsdl:input message="tns:GeneralFaultMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

Figure 4, WSDL portType Declarations for Participant and CoordinatorParticipant Roles

In order to perform the necessary interactions for normal participant-to-coordination interaction, two service roles are defined, with the following operations (message-exchanges):

- ParticipantCoordinator: this accepts the *setResponse* message. The endpoint address for the ParticipantCoordinator is returned to the Participant during the registration process (see below). The ParticipantRespondant address is propagated on all of these messages for call-back response messages.
- ParticipantRespondant: this accepts the *responseSet*, *unknownCoordinator*, *generalFault*, *protocolViolation* and *wrongState* messages.

Figure 5 illustrates the interactions between Participant and coordinator.

The ParticipantCoordinator can send the *setResponse* message because some coordination protocols will allow participants to make autonomous decisions based upon their current state and assumptions about which notifications a coordinator may send them. This operation is called to notify the coordinator identified in the associated context of the response (the AssertionType) from the Participant. It is valid for the AssertionType parameter to be nil. The identity of the message (the message URI) that triggered the Participant and the Participant identity are also returned, as is a QName which represents some coordination-specific response; this is to allow Participants to asynchronously send responses to messages that the ActivityCoordinator has not yet (and may never) send: the coordinator is required to record both sets of data until the next coordination point where it can determine, using the AssertionType provided by the Participant, whether or not it should send coordination messages to the Participant. If the Participant sent a response to a message the coordinator decided not to generate (e.g., it sent PREPARED assuming the coordinator would prepare when in fact the coordinator rolls back), then it is up to the implementation to determine what to do. Obviously if the Participant is allowed to make an asynchronous response then the protocol should be able to deal with this eventuality.

Upon successfully receiving and recording the message, the coordinator will call-back with the *responseSet* message. If the identity of the coordinator is invalid, then the *unknownCoordinator* message will be sent to the ParticipantRespondant. If the message sent by the Participant is incompatible with the current state of the coordinator, the coordinator will send the *protocolViolation* message; if the coordinator refuses to accept the message from the Participant then the *wrongState* message will be sent to the ParticipantRespondant.

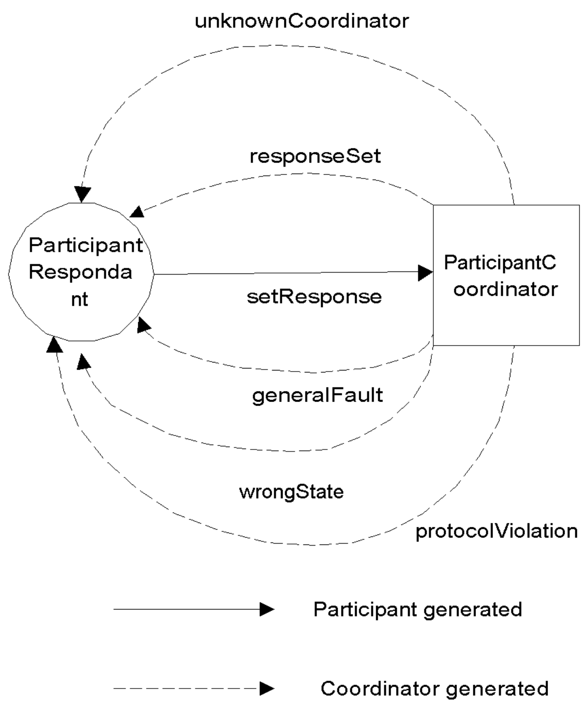

Figure 5, Participant-to-coordinator interactions.

The ParticipantCoordinator and ParticipantRespondant roles are presented in WSDL in Figure 6.

```
<wsdl:portType name="ParticipantCoordinatorPortType">
  <wsdl:operation name="setResponse">
    <wsdl:input message="tns:SetResponseMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="ParticipantRespondantPortType">
  <wsdl:operation name="responseSet">
    <wsdl:input message="tns:ResponseSetMessage"/>
  </wsdl:operation>
  <wsdl:operation name="unknownCoordinator">
    <wsdl:input message="tns:UnknownCoordinatorFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="generalFault">
    <wsdl:input message="tns:GeneralFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="protocolViolation">
    <wsdl:input message="asw:ProtocolViolationFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="wrongState">
    <wsdl:input message="asw:WrongStateFaultMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

Figure 6, WSDL portType Declarations for ParticipantCoordinator and ParticipantRespondant Roles.

4.2 Qualifiers

Qualifiers are a feature of WS-CF that allows additional protocol specific and business specific information to be exchanged by participating services. Typically qualifiers are used by participants when enrolling with a coordinator to augment the enrolment or un-enrolment operations (the addParticipant and removeParticipant operations) and thus enhance the coordination protocol. For example, when enlisting a participant with a transaction, it is possible to specify a caveat on enrolment via a suitable qualifier, such that the coordinator knows that the participant will cancel the work if it does not hear from the coordinator within 24 hours. The schema fragment for WS-CF qualifiers is shown in Figure 7.

```
<xs:complexType name="QualifierType">
  <xs:sequence>
    <xs:element name="qualifier-name" type="xs:string"/>
    <xs:any namespace="##any" processContents="lax" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

Figure 7, Qualifier XML Schema Type

4.3 Coordinator

An *activity coordinator* is associated with each activity; this happens implicitly through the appropriate Activity Lifecycle Service (ALS) that is enlisted with the CTX Service framework. This ALS is informed when the activity starts (and in which case it may create a new coordinator) and when it is completing (and in which case it will execute the coordination protocol across the registered participants). When a message is sent by the activity (e.g., at termination time), the coordinator's role is to forward this to all registered Participants and to deal with the outcomes generated by the Participants.

The protocol that the coordinator implementation uses will depend upon the type of activity, application or service using the coordination service. For example, if the coordination service is being used for within an extended transaction infrastructure, then one protocol implementation will not be sufficient. For example, if Saga model is in use then a compensation message may be required to be sent to Participants if a failure has happened, whereas a coordinator for a strict transactional model may be required to send a message informing participants to rollback.

How an ALS for a specific coordination protocol(s) is located and ultimately registered with the CTX Service is out of scope of this specification. An ALS may identify the type of coordination protocol it supports via the ALS *identify* message, but other deployment specific mechanisms may be used.

It is further envisaged that the Coordinator implementation can be a common/generic infrastructure component that is neutral to a particular Coordination Service implementation. The Coordinator is merely the registration point for interested participants of an activity. Obviously each such registration point will be required to publish the protocol it uses when performing coordination using the schema shown earlier.

A Coordination Service implementation provides:

- Transmission of coordination specific messages over SOAP requires a publish/subscribe or broadcast message interaction pattern;
- Support for the Participant service interface between CTX Service and Participant.

All operations on the coordinator service are implicitly associated with the current context, i.e., it is propagated to the coordinator service in order to identify which coordinator is to be operated on.

In the following sections we shall discuss the different coordinator interactions and their associated message exchanges.

4.3.1 Service-to-coordinator interactions

These interactions define how a service may enlist or delist a participant with the coordinator and perform other service-specific operations, and are illustrated in Figure 8. They are factored into two different roles:

- ServiceCoordinator: this accepts the *addParticipant*, *removeParticipant*, *getQualifiers* and *getParentCoordinator* messages. All messages contain the ServiceRespondant endpoint for call-back messages. It is this call-back address that is referenced in the extended context which is propagated between application services. The ServiceRespondant endpoint address is propagated on all of these messages.
- ServiceRespondant: this accepts the *participantAdded*, *participantRemoved*, *qualifiers*, *parentCoordinator*, *generalFault*, *unknownCoordinator*, *wrongState*, *duplicateParticipant*, *invalidProtocol*, *invalidParticipant*, *participantNotFound* messages.

addParticipant

This message is sent to the coordinator in order to register the specified Participant with the ActivityCoordinator identified in the context. If no coordinator can be located, then the *invalidCoordinator* message is sent to the ServiceRespondant.

The coordinator may support multiple sub-protocols (e.g., synchronizations that are executed prior to and after a two-phase commit protocol); in order to define with which protocol to enlist the participant, the *protocolType* URI is propagated in the message. If the protocol is not supported by this coordinator then the *invalidProtocol* message will be sent to the ServiceRespondant.

Upon success, the coordinator calls back to the ServiceRespondant with the *participantAdded* message, including in this message the ParticipantCoordinator address.

If the Activity has begun completion, or has already completed, then the *wrongState* message is sent.

If the same participant has been enrolled with the coordinator more than once and the coordination protocol does not allow this, then the *duplicateParticipant* message is sent to the ServiceRespondant.

If the participant is invalid within the scope of the coordinator, the *invalidParticipant* message is sent to the ServiceRespondant.

removeParticipant

This message causes the coordinator to remove the specified Participant from the ActivityCoordinator identifier in the associated context. If the Participant has not previously been registered with the coordinator for the specified coordination protocol, then it will send the *participantNotFound* message to the ServiceRespondant.

If no coordinator can be located, then the *invalidCoordinator* message is sent to the ServiceRespondant.

Removal of a participant need not be supported by the specific coordination implementation and obviously it may also be dependant upon where in the protocol the coordinator is as to whether it will allow the participant to be removed.

If the Activity has begun completion, or has completed, then the *wrongState* message is sent.

getParentCoordinator

This message causes the address of the parent coordinator of the coordinator referenced in the associated context to be sent to the ServiceRespondant via the parentCoordinator message. If there is no parent (i.e., this coordinator is top-level), then an empty address will be sent.

If no coordinator can be located, then the *invalidCoordinator* message is sent to the ServiceRespondant.

getQualifiers

This message causes the coordinator service to return the list of all qualifiers currently registered with it via the qualifiers message on the ServiceRespondant. If no coordinator can be located, then the *invalidCoordinator* message is sent to the ServiceRespondant.

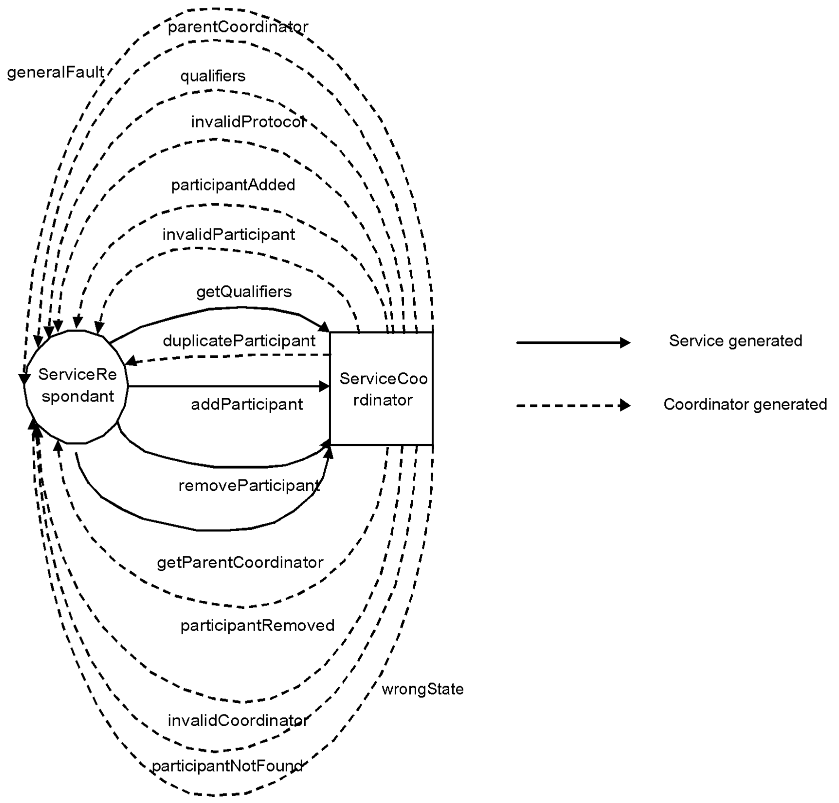

Figure 8, Service-to-coordinator interactions.

The ServiceRespondant and ServiceCoordinator roles are elucidated in WSDL form in Figure 9.

```
<wsdl:portType name="ServiceCoordinatorPortType">
  <wsdl:operation name="addParticipant">
    <wsdl:input message="tns:AddParticipantMessage"/>
  </wsdl:operation>
  <wsdl:operation name="removeParticipant">
    <wsdl:input message="tns:RemoveParticipantMessage"/>
  </wsdl:operation>
  <wsdl:operation name="getQualifiers">
    <wsdl:input message="tns:GetQualifiersMessage"/>
  </wsdl:operation>
  <wsdl:operation name="getParentCoordinator">
    <wsdl:input message="tns:GetParentCoordinatorMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="ServiceRespondantPortType">
  <wsdl:operation name="participantAdded">
    <wsdl:input message="tns:ParticipantAddedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="participantRemoved">
    <wsdl:input message="tns:ParticipantRemovedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="qualifiers">
    <wsdl:input message="tns:QualifiersMessage"/>
  </wsdl:operation>
  <wsdl:operation name="parentCoordinator">
    <wsdl:input message="tns:ParentCoordinatorMessage"/>
  </wsdl:operation>
  <wsdl:operation name="generalFault">
    <wsdl:input message="tns:GeneralFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="unknownCoordinator">
    <wsdl:input message="tns:UnknownCoordinatorFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="wrongState">
    <wsdl:input message="asw:WrongStateFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="duplicateParticipant">
    <wsdl:input message="tns:DuplicateParticipantFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="invalidProtocol">
    <wsdl:input message="tns:InvalidProtocolFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="invalidParticipant">
    <wsdl:input message="tns:InvalidParticipantMessage"/>
  </wsdl:operation>
  <wsdl:operation name="participantNotFound">
    <wsdl:input message="tns:ParticipantNotFoundFaultMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

Figure 9, WSDL portType Declarations for ServiceRespondant and ServiceCoordinator Roles.

4.3.2 Client-to-coordinator interactions

These interactions (illustrated in Figure 10) essentially define how a client (user) of the coordinator service can obtain the status of the coordinator or ask it to perform coordination. They are factored into two different services:

- ClientCoordinator: supports the *coordinate* and *getStatus* messages. All messages contain the ClientRespondant endpoint for call-back results. The ClientRespondant endpoint address is propagated on all of these messages.
- ClientRespondant: supports the *coordinated*, *status*, *wrongState*, *notCoordinated*, *protocolViolation*, *invalidCoordinator*, *invalidActivity* and *generalFault* messages.

coordinate

If the coordination protocol supports it then the coordinator will execute a particular coordination protocol (specified by a protocol URI) on the currently enlisted participants, upon receiving the *coordinate* message at any time *prior* to the termination of the coordination scope. This message instructs the ActivityCoordinator to send protocol messages to all of the registered Participants; since the coordinator may be invoked multiple times during the lifetime of an activity, it is possible that different protocol messages may be sent each time *coordinate* is called. Once the Participants have processed the messages and returned outcomes, it is up to the ActivityCoordinator to consolidate these individual outcomes into a single result, which is sent to the ClientRespondant via the *coordinated* message.

If there is no Activity associated with the context then the *invalidCoordinator* message will be generated.

Because this operation can be used to cause messages to be sent to Participants at times other than when the Activity completes, the implementation of the coordinator must ensure that such messages clearly identify that the Activity is not completing. If the Activity has begun completion, or has completed, then the *invalidActivity* message is sent to the ClientRespondant.

The coordinator may also send the *protocolViolation* or *wrongState* messages to the ClientRespondant to indicate appropriate error conditions that may occur while executing the coordination protocol.

The *notCoordinated* response is used to indicate that the coordinator (and hence coordination protocol) does not allow coordination to occur at any time other than the termination of the activity. Other, protocol specific errors are expected to be returned as data encoded within the AssertionType.

getStatus

The status of the coordinator may be obtained by sending the *getStatus* message to the coordinator. The status, which may be one of the status values specified by the CTX Service, or may be specific to the coordination protocol, identified by its QName, is returned to the ClientRespondant via the *status* message.

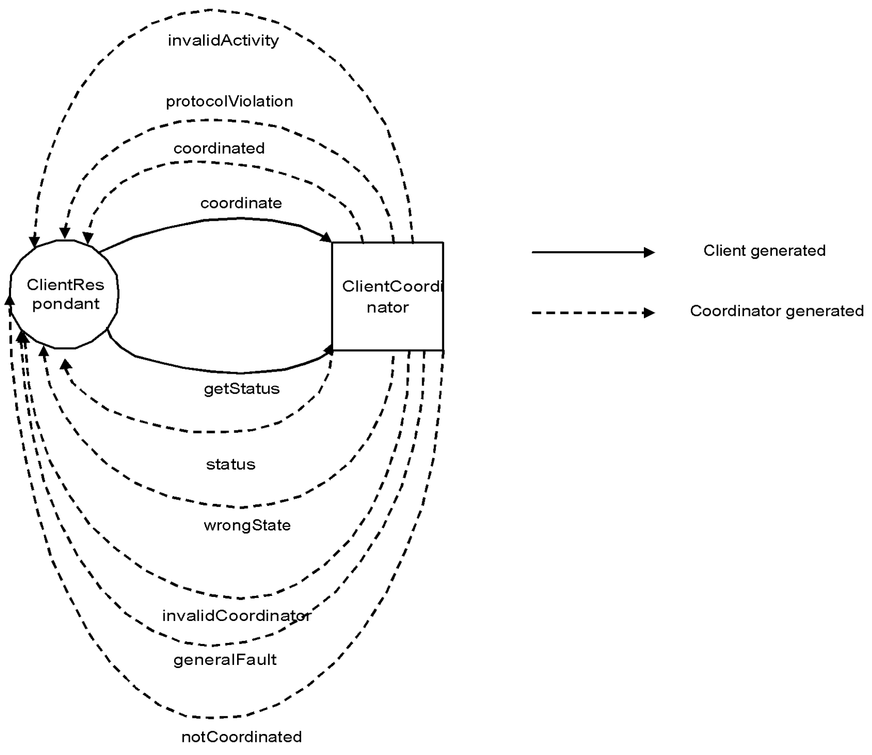

Figure 10, Client-to-coordinator interactions.

The ClientRespondant and ClientCoordinator roles are shown in WSDL form in Figure 11.

```
<wsdl:portType name="ClientCoordinatorPortType">
  <wsdl:operation name="coordinate">
    <wsdl:input message="tns:CoordinateMessage"/>
  </wsdl:operation>
  <wsdl:operation name="getStatus">
    <wsdl:input message="tns:GetStatusMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="ClientRespondantPortType">
  <wsdl:operation name="status">
    <wsdl:input message="tns:StatusMessage"/>
  </wsdl:operation>
  <wsdl:operation name="coordinated">
    <wsdl:input message="tns:CoordinatedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="notCoordinated">
    <wsdl:input message="tns:NotCoordinatedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="wrongState">
    <wsdl:input message="asw:WrongStateFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="protocolViolation">
    <wsdl:input message="asw:ProtocolViolationFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="invalidCoordinator">
    <wsdl:input message="tns:InvalidCoordinatorFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="invalidActivity">
    <wsdl:input message="tns:InvalidActivityFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="generalFault">
    <wsdl:input message="tns:GeneralFaultMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

Figure 11, WSDL portType Declarations for ClientRespondant and ClientCoordinator Roles

Context enhancement

In order to perform coordination, it is necessary for the participants to be enrolled with coordinators. In a distributed environment, this requires information about the coordinator (essentially its network endpoint) to be available to remote participants. The CTX Service is already responsible for propagating basic context information between distributed activities. As we have seen, the information contained within this basic activity context is simply the unique activity identity. However, it has been designed to be extensible such that additional, service-specific information may be added to the context via Activity Lifecycle Services. In the case of the relevant coordination lifecycle service, this information is the hierarchy of coordinator references.

```
<xs:complexType name="ContextType">
  <xs:complexContent>
    <xs:extension base="wsc4c:ContextType">
      <xs:sequence>
      <xs:element name="protocol-reference" type="tns:ProtocolReferenceType"/>
      <xs:element name="coordinator-reference" type="tns:CoordinatorReferenceType"
        maxOccurs="unbounded"/>
      <xs:any namespace="##any" processContents="lax" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 12, WS-CF ContextType derives from the WS-CTX ContextType.

The XML below shows an example of a coordination context for a coordinator implementation of a two-phase completion protocol.

```
<context xmlns="http://www.webservicestransactions.org/schemas/wsc4c/2003/03"
    timeout="100">
    <context-identifier>
      http://www.webservicestransactions.org/wsc4c/abcdef:012345
    </context-identifier>
    <activity-service>
      http://www.webservicestransactions.org/wsc4c/service
    </activity-service>
    <type>
      http://www.webservicestransactions.org/wsc4c/context/type1
    </type>
    <activity-list>
      <service>http://www.webservicestransactions.org/service1</service>
      <service>http://www.webservicestransactions.org/service2</service>
    </activity-list>
    <child-contexts>
      <child-context timeout="200">
        <context-identifier>
          http://www.webservicestransactions.org/wsc4c/5e4f2218b
        </context-identifier>
       <activity-service>
         http://www.webservicestransactions.org/wsc4c/service
       </activity-service>
    <type>http://www.webservicestransactions.org/wsc4c/context/type1</type>
    <activity-list mustUnderstand="true" mustPropagate="true">
      <service>http://www.webservicestransactions.org/service3</service>
      <service>http://www.webservicestransactions.org/service4</service>
    </activity-list>
    </child-context>
  </child-contexts>
  <protocol-reference protocolType="http://www.webservicestransactions.org/some-ref"/>
  <coordinator-reference coordinator="http://www.webservicestransactions.org/coord"
    activityIdentity="http://www.webservicestransactions.org/some-activity"/>
</context>
```

4.4 Interposition

Consider the situation depicted in Figure 13, where there is a coordinator and three participants. If we assume that each of these participants is on a different machine to the coordinator and each other then each of the lines connecting the coordinator to the participants also represents the invocations from the coordinator to the participants and vice versa.

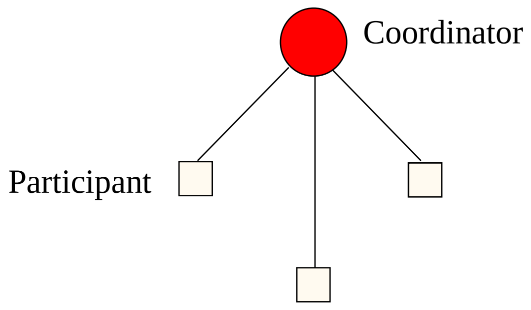

Figure 13, Coordinator-participant distributed interactions.

The overhead involved in making these distributed invocations will depend upon a number of factors, including how congested the network is, the load on the respective machines and the size of the coordination domain In addition, as the number of participants increase, so does the overhead involved in the coordinator executing the coordination protocol.

A common approach to ameliorate this overhead is to first recognize the fact that as far as a coordinator is concerned it does not matter what the participant implementation is: although one participant may interact with a database to commit a transaction, another may just as readily be responsible for forwarding the coordinators' messages to a number of databases: essentially acting as a coordinator itself, as shown in Figure 14.

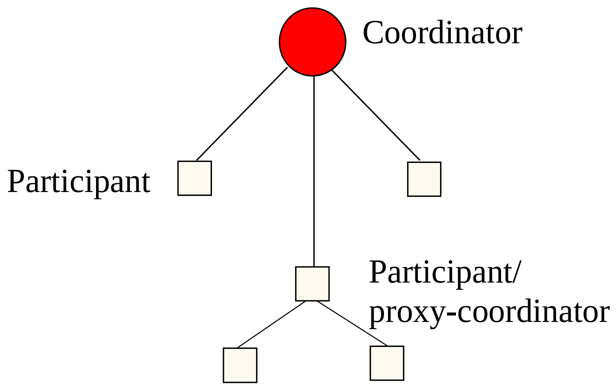

Figure 14, Participant coordinator.

In this case, the participant is acting like a proxy for the coordinator (the root coordinator): in the example, the proxy coordinator is responsible for interacting with the two participants when it receives an invocation from the coordinator and collating their responses (and it's own) for the coordinator. As far as the participants are concerned they are invoked by a coordinator, whereas as far as the root coordinator is concerned it only sees participants.

This technique of using proxy coordinators (or subordinate (sub-) coordinators) is known as *interposition*. Each domain that imports a context may create a subordinate coordinator that enrolls with the imported coordinator as though it were a participant. Interposition obviously requires the domain to use a different context when communicating with services and participants within the domain since at the very least the coordinator endpoint will be different. Any participants that are required to enroll with the coordinated activity within this domain actually enroll with the subordinate coordinator. In a large distributed application, a tree of coordinators and participants may be created, as illustrated in Figure 15. WS-CF does not mandate that interposition is supported by an implementation.

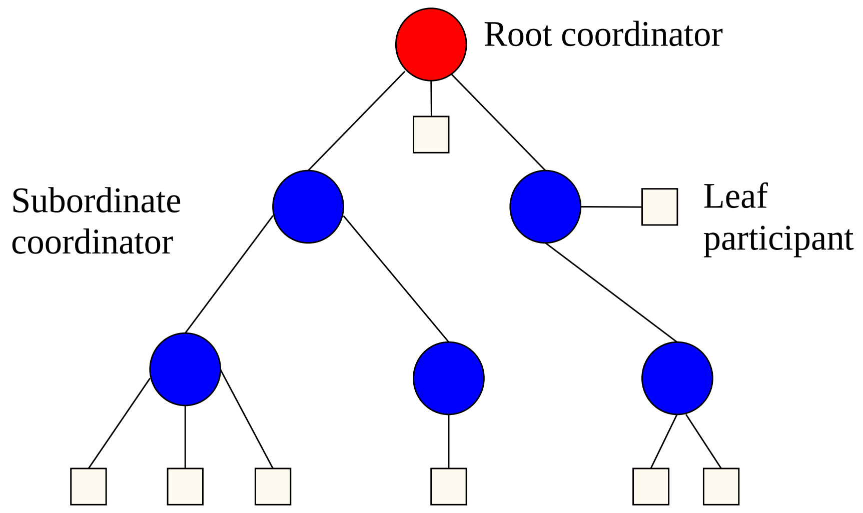

Figure 15, Interposition.

Because a subordinate coordinator must execute the coordination protocol on its enlisted participants, it must have its own log and corresponding failure recovery subsystem. The subordinate must record sufficient recovery information for any work it may do as a participant *and* additional recovery information for its role as a coordinator.

State management and recovery

It is inherently complex to recover applications after failures (e.g., machine crashes). For example, the states of objects in use prior to the failure may be corrupt. The advantage of using transactions to control operations on persistent objects is that transaction systems ensure the consistency of the objects, regardless of whether or not failures occur. A transaction system guarantees that regardless of (non-catastrophic) failures, all transactions that were in flight when the failure occurred will either be committed or rolled back, making permanent or undoing any changes to objects.

Rather than mandate a particular means by which objects should make themselves persistent, many transaction systems simply state the requirements they place on such objects if they are to be made recoverable, and leave it up to the object implementers to determine the best strategy for their object's persistence. The transaction system itself will have to make sufficient information persistent such that, in the event of a failure and subsequent recovery, it can tell these objects whether to commit any state changes or roll them back. However, it is typically not responsible for the application object's persistence.

In a similar way, the WS-CF specification does not mandate a specific persistence and recovery mechanism. Rather it states what the requirements are on such a service in the event of a failure, and leaves it to individual implementers to determine their own recovery mechanisms. In a distributed application, where an individual activity may run on different implementations of the WS-CF during its lifetime, recovery is the responsibility of these different implementations. Each implementation may perform recovery in a completely different manner, forming *recovery domains*.

Note, failure recovery semantics are strongly tied to the protocol that the coordinator supports. As such, information about for how long a coordinator must remember failures and their participants cannot be mandated by this specification. It is important that the contract that exists between coordinator and participant is defined by the implementer of the coordination protocol, especially in the case of failures. It is this contract that will be used by both the coordinator and participant to interpret responses to the recovery protocol.

Unlike in a traditional transactional system, where crash recovery mechanisms are only responsible for guaranteeing consistency of object data, applications that use Coordination Service's will typically also require the ability to recover the activity structure that was present at the time of the failure, enabling the application to progress onwards.

Some of the recovery requirements are outlined below:

- *application logic*: the logic required to drive the activities during normal runtime is required during recovery in order to drive any in-flight activities to application specific consistency. Since it is the application level that imposes meaning on Participants and messages, it is predominately the application that is responsible for driving recovery.
- *application object consistency*: the states of all application objects must be returned to some form of application specific consistency after a failure.

The following roles are defined to assist in recovery; the message interactions are shown in Figure 16:

- RecoveryCoordinator: this service is used to drive recovery on behalf of a participant. It supports the *recover* and *getStatus* messages. The RecoveryParticipant endpoint address is propagated on all of these messages for call-back results.
- RecoveryParticipant: this service is used to return the recovery information to a recovering participant via call-backs. It supports the *recovered, status, unknownCoordinator, wrongState* and *generalFault* messages.

recover

This operation is used by participants that have previously successfully registered with a coordinator. When a participant fails and subsequently recovers it may not be able to recover at the same address that it used to enlist with the coordinator. The *recover* operation allows the participant to inform that coordinator that the participant has moved from the original address to a new address. It may also be used to start recovery operations by the coordinator.

If successful, the *recoverResponse* message is sent to the RecoveryParticipant along with the current status of the transaction. This status may be used by the recovering participant to perform recovery, although this will depend upon the coordination protocol in use. For example, if the participant was enrolled in a presumed-abort transaction protocol and *recover* indicated that the transaction no longer exists, then the participant can cancel any work it may be controlling.

If the coordinator cannot be located, then the *unknownCoordinator* message is sent back.

If the status of the coordinator is such that recovery is not allowed at this time, the *wrongState* message is sent to the RecoveryParticipant by the coordinator.

getStatus

The status of the coordinator may be obtained by sending the *getStatus* message to the coordinator. The status, which may be one of the status values specified by the CTX Service, or may be specific to the coordination protocol, identified by its QName, is returned to the RecoveryParticipant via the *status* message.

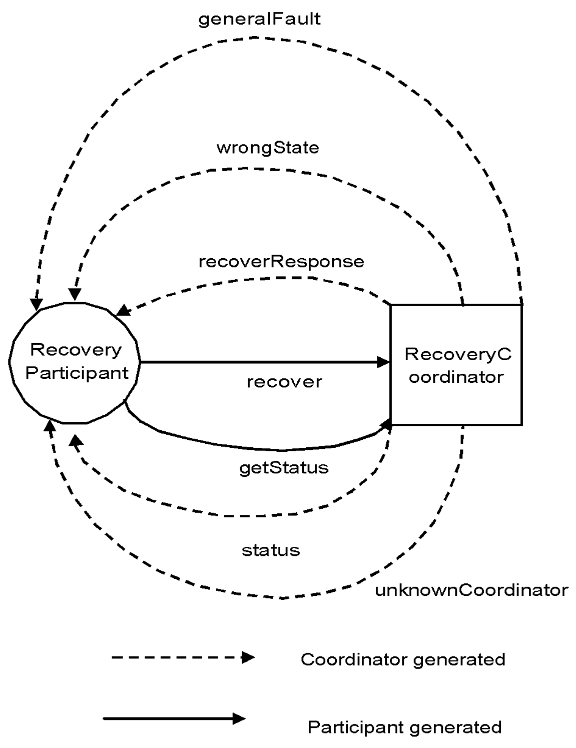

Figure 16, Participant recovery.

The RecoveryCoordinator and RecoveryParticipant interfaces are presented in Figure 17.

```
<wsdl:portType name="RecoveryCoordinatorPortType">
  <wsdl:operation name="recover">
  <wsdl:input message="tns:RecoverMessage"/>
  </wsdl:operation>
  <wsdl:operation name="getStatus">
  <wsdl:input message="tns:GetStatusMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="RecoveryParticipantPortType">
  <wsdl:operation name="recovered">
  <wsdl:input message="tns:RecoveredMessage"/>
  </wsdl:operation>
  <wsdl:operation name="status">
  <wsdl:input message="tns:StatusMessage"/>
  </wsdl:operation>
  <wsdl:operation name="unknownCoordinator">
  <wsdl:input message="tns:UnknownCoordinatorFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="wrongState">
  <wsdl:input message="asw:WrongStateFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="generalFault">
  <wsdl:input message="tns:GeneralFaultMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

Figure 17, WSDL portType Declarations for RecoveryParticipant and RecoveryCoordinator Roles

5. Roles & Responsibilities

With reference to Figure 18, the following section describes the roles and responsibilities specific to the WS-CF architecture.

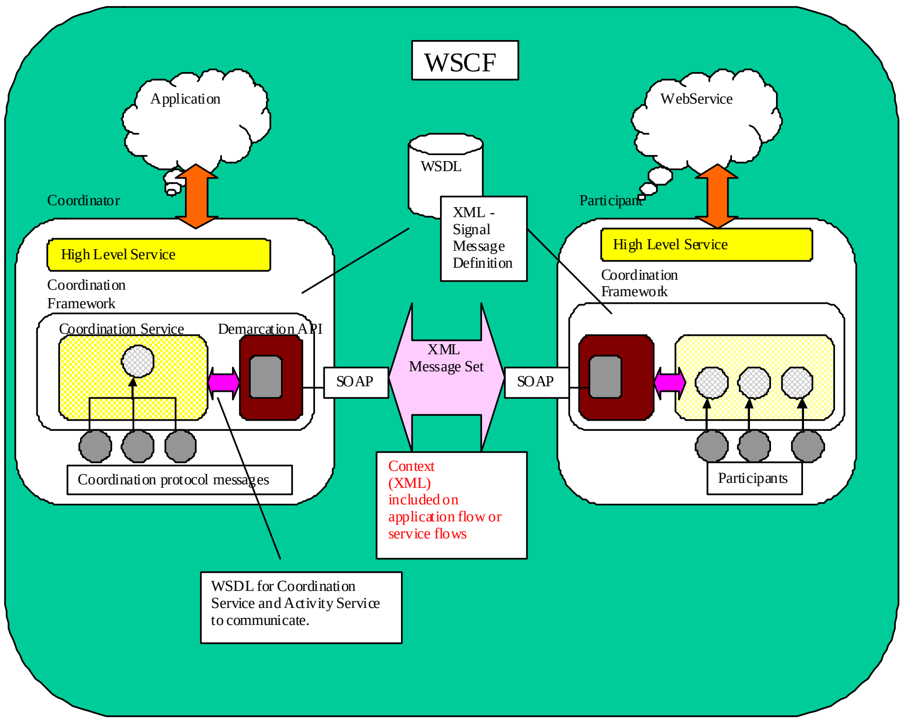

Figure 18, WS-CF components.

Coordination Service Activity Lifecycle Service provider

This Web service ties into the WS-CTX and allows the application to define the beginning and ending points of a coordinated activity and to direct the outcome. The scope of an activity becomes the scope of a coordinated interaction. The relationship between the ALS and the coordination service is not mandated by WS-CF.

Coordination Service Provider

The coordination service provider supplies an implementation of a completion processing facility that provides a means to orchestrate a number of tasks that have a common interest. Examples of such a coordination service include usage patterns for transactional activity (e.g., an OMG/OTS or Java/JTS Transaction Service implementation), extended/relaxed transactional activity (e.g., an OMG/OTS Additional Structuring Mechanism implementation to support other forms of processing such as long-running, collaboration or real-time activities) and other behaviors (including non-transactional groupings).

The definition of a coordination service supplies the following:

- Protocol: Defines the characteristics of a coordination service and the contracts & obligations for the participants of an activity.

Web Service Provider

The Web Service provider (or the resources associated with the Web Service) need to provide the following:

- A Participant implementation to respond to the coordination messages from a Coordination Service implementation. It is envisaged that Participants are interchangeable or pluggable to provide differing levels of Quality of Service depending on the Coordination Service utilized for an activity.
- Support the Participant API's (interface between CTX Service and Participant). It is the Participant that is the coordinated counterpart for the service that enlisted it with the coordinator. Obviously a service may act as a Participant, though this is not a requirement.

6. Example

Workflow systems with scripting facilities for expressing the composition of an activity (a business process) offer a flexible way of building application specific extended transactions. In this section we describe how WS-CF can be utilized for coordinating workflow activities. In this example, the coordinator starts new activities to perform units of work and eventually receives the results. As such, each Participant drives the lifecycle of an activity.

The coordinator-participant interaction protocol three messages, “start”, “start_ack”, “outcome”.

- *start*: the message is sent from a “parent” activity to a “child” activity, to indicate that the “child” activity should start (via an *AssertionType*). The message may contain additional information required to parameterize the starting of the activity (workflow task).
- *start_ack*: this AssertionType is sent from a “child” activity to a “parent” activity, as the result of a “start” message, to acknowledge that the “child” activity has started.
- *outcome*: this message is sent from a “child” activity to a “parent” activity, to indicate that the “child” activity has completed (via *setResponse*). The AssertionType may contain information about how the activity terminated, e.g., whether or not it completed successfully.

The interaction depicted in fig. 10 is activity *a* coordinating the parallel execution of *b* and *c* followed by *d*. Whenever a child activity is started the parent activity registers a Participant with it that is used to deliver the “outcome” to the parent.

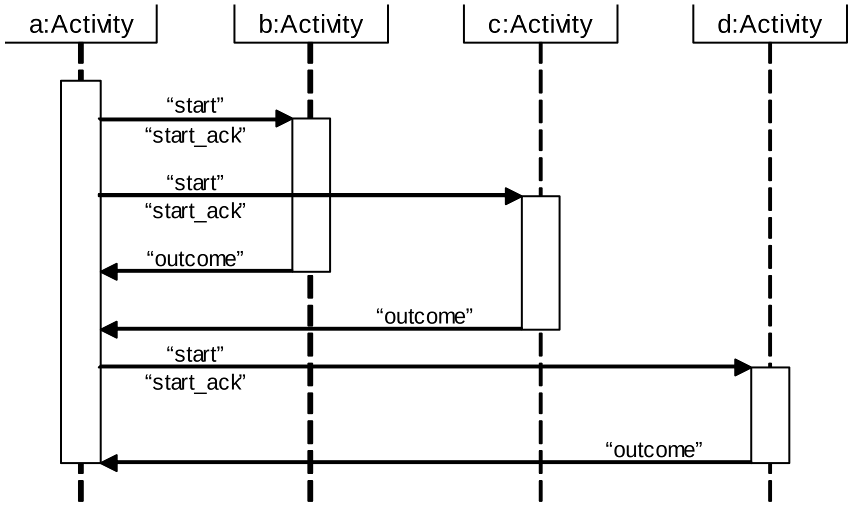

Figure 19, Workflow coordination.

7. XML Schema and WSDL Interfaces

The following sections describe the WSDL for the Web Services components of WS-CF as well as the XML for contexts, message formats etc.

The WSDL interfaces presented here are more concrete than those presented earlier in this document, and offer a straightforward SOAP binding (using document/literal) for transporting WS-CF messages.

7.1 XML Schema for WS-CF Messages

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.webservicestransactions.org/schemas/wscf/2003/03"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://www.webservicestransactions.org/schemas/wscf/2003/03"
xmlns:wsc4c="http://www.webservicestransactions.org/schemas/wsc4c/2003/03">
  <xs:import namespace="http://www.webservicestransactions.org/schemas/wsc4c/2003/03"
schemaLocation="../../WS-CTX/xml/wsc4c.xsd"/>
  <xs:complexType name="CoordinatorReferenceType">
    <xs:attribute name="coordinator" type="xs:anyURI" use="required"/>
    <xs:attribute name="activityIdentity" type="xs:anyURI" use="optional"/>
  </xs:complexType>
  <xs:complexType name="ProtocolReferenceType">
    <xs:attribute name="protocolType" type="xs:anyURI" use="required"/>
  </xs:complexType>
  <xs:complexType name="ContextType">
    <xs:complexContent>
      <xs:extension base="wsc4c:ContextType">
        <xs:sequence>
          <xs:element name="protocol-reference" type="tns:ProtocolReferenceType"/>
          <xs:element name="coordinator-reference" type="tns:CoordinatorReferenceType"
maxOccurs="unbounded"/>
          <xs:any namespace="##any" processContents="lax" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:simpleType name="StatusType">
    <xs:restriction base="wsc4c:StatusType"/>
  </xs:simpleType>
  <xs:element name="status" type="tns:StatusType" substitutionGroup="wsc4c:status"/>
  <xs:complexType name="CompletionStatusType">
    <xs:simpleContent>
      <xs:extension base="wsc4c:CompletionStatusType"/>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="QualifierType">
    <xs:sequence>
      <xs:element name="qualifier-name" type="xs:string"/>
      <xs:any namespace="##any" processContents="lax" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="qualifier" type="tns:QualifierType" abstract="true"/>
  <xs:complexType name="QualifiersType">
    <xs:sequence>
      <xs:element name="qualifier" type="tns:QualifierType" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="AssertionType">
    <xs:complexContent>
      <xs:extension base="wsc4c:AssertionType">
        <xs:sequence>
          <xs:element name="qualifiers" type="tns:QualifiersType" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="assertion" type="tns:AssertionType" abstract="true"/>
  <xs:complexType name="FaultType">
    <xs:complexContent>
      <xs:extension base="tns:AssertionType">
        <xs:sequence>
          <xs:element name="faultcode" type="xs:anyURI"/>
          <xs:element name="faultstring" type="xs:string" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="fault" type="tns:FaultType" abstract="true"/>
  <xs:element name="context" type="tns:ContextType" substitutionGroup="wsc4c:context"/>
  <xs:element name="get-identity" type="tns:AssertionType"
substitutionGroup="tns:assertion"/>
  <xs:element name="identity" type="tns:AssertionType"/>
  <xs:element name="set-response" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element name="assumed-message" type="tns:AssertionType"/>
            <xs:element name="response" type="tns:AssertionType"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="response-set" type="tns:AssertionType"
substitutionGroup="tns:assertion"/>
  <xs:element name="recover" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element name="old-participant" type="xs:anyURI"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="recovered" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element ref="tns:status"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="add-participant" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element name="participant" type="xs:anyURI"/>
            <xs:element name="protocol" type="xs:anyURI"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="participant-added" type="tns:AssertionType"
substitutionGroup="tns:assertion"/>
  <xs:element name="remove-participant" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element name="participant" type="xs:anyURI"/>
            <xs:element name="protocol" type="xs:anyURI"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="participant-removed" type="tns:AssertionType"
substitutionGroup="tns:assertion"/>
  <xs:element name="get-qualifiers" type="tns:AssertionType"
substitutionGroup="tns:assertion"/>
  <xs:element name="qualifiers" type="tns:AssertionType"
substitutionGroup="tns:assertion"/>
```

```
  <xs:element name="get-parent-coordinator" type="tns:AssertionType"
substitutionGroup="tns:assertion"/>
  <xs:element name="parent-coordinator">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element name="parent-coordinator" type="xs:anyURI"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="get-status" type="tns:AssertionType"
substitutionGroup="tns:assertion"/>
  <xs:element name="got-status" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element ref="tns:status"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="coordinate" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element name="sub-protocol" type="xs:anyURI"/>
            <xs:element name="completion-status" type="tns:CompletionStatusType"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="coordinated" type="tns:AssertionType"
substitutionGroup="tns:assertion"/>
  <xs:element name="unknown-coordinator-fault" substitutionGroup="tns:fault">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:FaultType">
          <xs:sequence>
            <xs:element name="coordinator" type="xs:anyURI"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="duplicate-participant-fault" substitutionGroup="tns:fault">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:FaultType">
          <xs:sequence>
            <xs:element name="duplicate-participant" type="xs:anyURI"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="participant-not-found-fault" substitutionGroup="tns:fault">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:FaultType">
          <xs:sequence>
            <xs:element name="participant" type="xs:anyURI"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="invalid-participant-fault" substitutionGroup="tns:fault">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:FaultType">
          <xs:sequence>
            <xs:element name="participant" type="xs:anyURI"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="invalid-protocol-fault" substitutionGroup="tns:fault">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:FaultType">
          <xs:sequence>
            <xs:element name="protocol" type="xs:anyURI"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="general-fault" substitutionGroup="tns:fault">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:FaultType">
          <xs:sequence>
            <xs:element name="faulting-actor" type="xs:anyURI"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

7.2 WSDL Interface for WS-CF Actors

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
targetNamespace="http://www.webservicestransactions.org/wsdl/wscf/2003/03"
xmlns:tns="http://www.webservicestransactions.org/wsdl/wscf/2003/03"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wscf="http://www.webservicestransactions.org/schemas/wscf/2003/03"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <wsdl:import namespace="http://www.webservicestransactions.org/schemas/wscf/2003/03"
location="wscf.xsd"/>
  <wsdl:message name="ContextMessage">
    <wsdl:part name="content" element="wscf:context"/>
  </wsdl:message>
  <wsdl:message name="GetIdentityMessage">
    <wsdl:part name="content" element="wscf:get-identity"/>
  </wsdl:message>
  <wsdl:message name="IdentityMessage">
    <wsdl:part name="content" element="wscf:identity"/>
  </wsdl:message>
  <wsdl:message name="SetResponseMessage">
    <wsdl:part name="content" element="wscf:set-response"/>
  </wsdl:message>
  <wsdl:message name="ResponseSetMessage">
    <wsdl:part name="content" element="wscf:response-set"/>
  </wsdl:message>
  <wsdl:message name="RecoverMessage">
    <wsdl:part name="content" element="wscf:recover"/>
  </wsdl:message>
  <wsdl:message name="RecoveredMessage">
    <wsdl:part name="content" element="wscf:recovered"/>
  </wsdl:message>
  <wsdl:message name="AddParticipantMessage">
    <wsdl:part name="content" element="wscf:add-participant"/>
  </wsdl:message>
  <wsdl:message name="ParticipantAddedMessage">
    <wsdl:part name="content" element="wscf:participant-added"/>
  </wsdl:message>
  <wsdl:message name="RemoveParticipantMessage">
    <wsdl:part name="content" element="wscf:remove-participant"/>
  </wsdl:message>
  <wsdl:message name="ParticipantRemovedMessage">
    <wsdl:part name="content" element="wscf:participant-removed"/>
  </wsdl:message>
  <wsdl:message name="GetQualifiersMessage">
    <wsdl:part name="content" element="wscf:get-qualifiers"/>
  </wsdl:message>
  <wsdl:message name="QualifiersMessage">
    <wsdl:part name="content" element="wscf:qualifiers"/>
  </wsdl:message>
  <wsdl:message name="GetParentCoordinatorMessage">
    <wsdl:part name="content" element="wscf:get-parent-coordinator"/>
  </wsdl:message>
  <wsdl:message name="ParentCoordinatorMessage">
    <wsdl:part name="content" element="wscf:parent-coordinator"/>
  </wsdl:message>
  <wsdl:message name="GetStatusMessage">
    <wsdl:part name="content" element="wscf:get-status"/>
  </wsdl:message>
  <wsdl:message name="StatusMessage">
    <wsdl:part name="content" element="wscf:got-status"/>
  </wsdl:message>
  <wsdl:message name="CoordinateMessage">
    <wsdl:part name="content" element="wscf:coordinate"/>
  </wsdl:message>
  <wsdl:message name="CoordinatedMessage">
    <wsdl:part name="content" element="wscf:coordinated"/>
  </wsdl:message>
  <wsdl:message name="UnknownCoordinatorFaultMessage">
    <wsdl:part name="content" element="wscf:unknown-coordinator-fault"/>
  </wsdl:message>
  <wsdl:message name="DuplicateParticipantFaultMessage">
    <wsdl:part name="content" element="wscf:duplicate-participant-fault"/>
  </wsdl:message>
  <wsdl:message name="ParticipantNotFoundFaultMessage">
    <wsdl:part name="content" element="wscf:participant-not-found-fault"/>
  </wsdl:message>
  <wsdl:message name="InvalidParticipantFaultMessage">
    <wsdl:part name="content" element="wscf:invalid-participant-fault"/>
  </wsdl:message>
  <wsdl:message name="InvalidProtocolFaultMessage">
```

```
    <wsdl:part name="content" element="wscf:invalid-protocol-fault"/>
  </wsdl:message>
  <wsdl:message name="GeneralFaultMessage">
    <wsdl:part name="content" element="wscf:general-fault"/>
  </wsdl:message>
  <wsdl:portType name="ParticipantPortType">
    <wsdl:operation name="getStatus">
      <wsdl:input message="tns:GetStatusMessage"/>
    </wsdl:operation>
    <wsdl:operation name="getIdentity">
      <wsdl:input message="tns:GetIdentityMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="CoordinatorParticipantPortType">
    <wsdl:operation name="status">
      <wsdl:input message="tns:StatusMessage"/>
    </wsdl:operation>
    <wsdl:operation name="Identity">
      <wsdl:input message="tns:IdentityMessage"/>
    </wsdl:operation>
    <wsdl:operation name="wrongState">
      <wsdl:input message="asw:WrongStateFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="generalFault">
      <wsdl:input message="tns:GeneralFaultMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="ParticipantCoordinatorPortType">
    <wsdl:operation name="setResponse">
      <wsdl:input message="tns:SetResponseMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="ParticipantRespondantPortType">
    <wsdl:operation name="responseSet">
      <wsdl:input message="tns:ResponseSetMessage"/>
    </wsdl:operation>
    <wsdl:operation name="unknownCoordinator">
      <wsdl:input message="tns:UnknownCoordinatorFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="generalFault">
      <wsdl:input message="tns:GeneralFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="protocolViolation">
      <wsdl:input message="asw:ProtocolViolationFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="wrongState">
      <wsdl:input message="asw:WrongStateFaultMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="ServiceCoordinatorPortType">
    <wsdl:operation name="addParticipant">
      <wsdl:input message="tns:AddParticipantMessage"/>
    </wsdl:operation>
    <wsdl:operation name="removeParticipant">
      <wsdl:input message="tns:RemoveParticipantMessage"/>
    </wsdl:operation>
    <wsdl:operation name="getQualifiers">
      <wsdl:input message="tns:GetQualifiersMessage"/>
    </wsdl:operation>
    <wsdl:operation name="getParentCoordinator">
      <wsdl:input message="tns:GetParentCoordinatorMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="ServiceRespondantPortType">
    <wsdl:operation name="participantAdded">
      <wsdl:input message="tns:ParticipantAddedMessage"/>
    </wsdl:operation>
    <wsdl:operation name="participantRemoved">
      <wsdl:input message="tns:ParticipantRemovedMessage"/>
    </wsdl:operation>
    <wsdl:operation name="qualifiers">
      <wsdl:input message="tns:QualifiersMessage"/>
    </wsdl:operation>
    <wsdl:operation name="parentCoordinator">
      <wsdl:input message="tns:ParentCoordinatorMessage"/>
    </wsdl:operation>
    <wsdl:operation name="generalFault">
      <wsdl:input message="tns:GeneralFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="unknownCoordinator">
      <wsdl:input message="tns:UnknownCoordinatorFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="wrongState">
      <wsdl:input message="asw:WrongStateFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="duplicateParticipant">
      <wsdl:input message="tns:DuplicateParticipantFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="invalidProtocol">
      <wsdl:input message="tns:InvalidProtocolFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="invalidParticipant">
      <wsdl:input message="tns:InvalidParticipantMessage"/>
    </wsdl:operation>
    <wsdl:operation name="participantNotFound">
      <wsdl:input message="tns:ParticipantNotFoundFaultMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="ClientCoordinatorPortType">
    <wsdl:operation name="coordinate">
      <wsdl:input message="tns:CoordinateMessage"/>
    </wsdl:operation>
    <wsdl:operation name="getStatus">
      <wsdl:input message="tns:GetStatusMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="ClientRespondantPortType">
    <wsdl:operation name="status">
      <wsdl:input message="tns:StatusMessage"/>
    </wsdl:operation>
    <wsdl:operation name="coordinated">
      <wsdl:input message="tns:CoordinatedMessage"/>
    </wsdl:operation>
    <wsdl:operation name="notCoordinated">
      <wsdl:input message="tns:NotCoordinatedMessage"/>
    </wsdl:operation>
    <wsdl:operation name="wrongState">
      <wsdl:input message="asw:WrongStateFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="protocolViolation">
      <wsdl:input message="asw:ProtocolViolationFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="invalidCoordinator">
      <wsdl:input message="tns:InvalidCoordinatorFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="invalidActivity">
      <wsdl:input message="tns:InvalidActivityFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="generalFault">
      <wsdl:input message="tns:GeneralFaultMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="RecoveryCoordinatorPortType">
    <wsdl:operation name="recover">
      <wsdl:input message="tns:RecoverMessage"/>
    </wsdl:operation>
    <wsdl:operation name="getStatus">
      <wsdl:input message="tns:GetStatusMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="RecoveryParticipantPortType">
```

```
    <wsdl:operation name="recovered">
      <wsdl:input message="tns:RecoveredMessage"/>
    </wsdl:operation>
    <wsdl:operation name="status">
      <wsdl:input message="tns:StatusMessage"/>
    </wsdl:operation>
    <wsdl:operation name="unknownCoordinator">
      <wsdl:input message="tns:UnknownCoordinatorFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="wrongState">
      <wsdl:input message="asw:WrongStateFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="generalFault">
      <wsdl:input message="tns:GeneralFaultMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="ParticipantPortTypeSOAPBinding" type="tns:ParticipantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="getStatus">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/getStatus"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="getIdentity">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/getIdentity"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="CoordinatorParticipantPortTypeSOAPBinding"
type="tns:CoordinatorParticipantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="status">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/status"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="Identity">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/Identity"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="wrongState">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/wrongState"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="generalFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/generalFault"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="ParticipantCoordinatorPortTypeSOAPBinding"
type="tns:ParticipantCoordinatorPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="setResponse">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/setResponse"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="ParticipantRespondantPortTypeSOAPBinding"
type="tns:ParticipantRespondantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="responseSet">
```

```
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/responseSet"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="unknownCoordinator">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/unknownCoordinator"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="generalFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/generalFault"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="protocolViolation">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/protocolViolation"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="wrongState">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/wrongState"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="ServiceCoordinatorPortTypeSOAPBinding"
type="tns:ServiceCoordinatorPortType">
```

```
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="addParticipant">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/addParticipant"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="removeParticipant">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/removeParticipant"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="getQualifiers">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/getQualifiers"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="getParentCoordinator">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/getParentCoordinator"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="ServiceRespondantPortTypeSOAPBinding"
type="tns:ServiceRespondantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="participantAdded">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/participantAdded"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="participantRemoved">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/participantRemoved"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="qualifiers">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/qualifiers"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="parentCoordinator">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/parentCoordinator"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="generalFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/generalFault"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="unknownCoordinator">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/unknownCoordinator"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="wrongState">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/wrongState"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="duplicateParticipant">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/duplicateParticipant"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="invalidProtocol">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/invalidProtocol"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="invalidParticipant">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/invalidParticipant"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="participantNotFound">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/participantNotFound"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="ClientCoordinatorPortTypeSOAPBinding"
type="tns:ClientCoordinatorPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="coordinate">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/coordinate"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="getStatus">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/getStatus"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="ClientRespondantPortTypeSOAPBinding"
type="tns:ClientRespondantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="status">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/status"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="coordinated">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/coordinated"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="notCoordinated">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/notCoordinated"
style="document"/>
```

```
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="wrongState">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/wrongState"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="protocolViolation">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/protocolViolation"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="invalidCoordinator">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/invalidCoordinator"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="invalidActivity">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/invalidActivity"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="generalFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/generalFault"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="RecoveryCoordinatorPortTypeSOAPBinding"
type="tns:RecoveryCoordinatorPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="recover">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/recover"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="getStatus">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/getStatus"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="RecoveryParticipantPortTypeSOAPBinding"
type="tns:RecoveryParticipantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="recovered">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/recovered"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="status">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/status"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="unknownCoordinator">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/unknownCoordinator"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="wrongState">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/wrongState"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="generalFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wscf/2003/03/generalFault"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

7.3 Issues

Other issues that will need to be considered when implementing many business transactions include:

1) Security and confidentiality: any business transaction involving buying or selling items, whether they be hotel rooms or newspapers, requires guarantees that the buyer/seller is who they appear to be, and that no one can “snoop” the connection and obtain information they are not entitled to.

2) Audit trail: maintaining a log of the actions performed during a business transaction can be useful for a number of reasons, not least that of non-repudiation in the case of legal action.

3) Protocol completeness guarantee: even in the presence of failures, the correctness guarantee for the application relies upon the structure of the application activity being followed. The information about which activity to invoke when and under what circumstances must reside in, for example, a highly available repository, such that

failure of the original “controller” (that entity which was responsible for parsing and driving the activities) does not cause the activity to stop, or for branches of it to be ignored.

4) Quality of service: some Web Services may support different types of extended transaction model as well as different communication protocols. The selection of which model to use may depend upon quality of service requirements.

How these fit into the WS-CF will be one of the areas of future research and development.

8. References

[1] OMG, Additional Structuring Mechanisms for the OTS Specification, September 2000, document orbos/2000-04-02.

[2] WSDL 1.1 Specification. See http://www.w3.org/TR/wsdl

9. Acknowledgements

The authors would like to thank the following people for their contributions to this specification:

Dave Ingham, Arjuna Technologies Ltd.

Barry Hodgson, Arjuna Technologies Ltd.

Goran Olsson, Oracle Corporation.

Nickolas Kavantzas, Oracle Corporation.

Aniruddha Thankur, Oracle Corporation.