Web Services Coordination Framework Specification (WS-CF)

Committee draft version 0.2

22 December 2004

Deleted: 20 December 2004

Inserted: 20 December 2004

Deleted: 7 December 2004

Abstract

WS-CAF provides a set of modular and composable service definitions to facilitate the construction of applications that combine multiple services together in composite applications. The fundamental capability offered by the WS-Coordination Framework specification is the ability to register a web service as a participant in some kind of domain specific function. An example scenario may be to register with a publication-subscription topic to receive a stream of messages asynchronously. While it is expected that the vast majority of protocols will involve some form of signaling to registered services via SOAP messages, this signaling is not a part of the model itself. Monitoring protocols, for example, may express interest in participation is some interaction semantic without any subsequent signaling to registered services; messaging protocols may use an optimized channel based on a native MOM protocol for message distribution.

WS-Context provides a late binding session model for the web services environment. SOAP messages that are to be processed within the scope of an activity contain Context headers, uniquely identifying a single activity. WS-Coordination Framework extends the session model for protocols that require group membership paradigms by defining a “registration context”. The “registration context” extends the basic context type and provides a Web service reference to a “registration” endpoint. Registration in the context of an activity adds the registered service to an activity group. Membership in the group may be used to drive some group specific protocol (e.g. data replication) over the lifetime of the activity group or may be used to coordinate signals associated with a termination protocol (e.g., two phase commit). The purpose and semantics of activity group membership are protocol specific.

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. Coordination protocols may be layered on WS-Coordination Framework.

Comment: Intro and abstract both need to be rewritten for clarity and topic order (i.e. ensure topics are presented in order of importance). This is an editorial issue, and may depend upon the resolution of some of the technical issues.

Table of contents

Deleted: 7
Deleted: 8
Deleted: 10
Deleted: 11
Deleted: 12
Deleted: 12
Deleted: 13
Deleted: 14
Deleted: 14
Deleted: 15
Deleted: 16
Deleted: 17
Deleted: 18
Deleted: 19
Deleted: 21

1 Note on terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119 [2].

Namespace URIs of the general form "some-URI" represents some application-dependent or context-dependent URI as defined in RFC 2396 [3].

1.1 Namespace

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

```
http://docs.oasis-open.org/wscaf/2004/09/wscf
```

1.1.1 Prefix Namespace

Prefix	Namespace
wscf	**http://docs.oasis-open.org/wscaf/2004/09/wscf**
wsctx	**http://docs.oasis-open.org/wscaf/2004/09/wsctx**
ref	**http://docs.oasisopen.org/wsrm/2004/06/reference-1.1**
wsdl	**http://schemas.xmlsoap.org/wsdl/**
xsd	**http://www.w3.org/2001/XMLSchema**
wsu	**http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd**
tns	**targetNamespace**

Comment: Need to agree on this.

1.2 Referencing Specifications

One or more other specifications, such as (but not limited to) WS-TXM may reference the WS-CF specification. The usage of optional items in WS-CF is typically determined by the requirements of such as referencing specification.

Referencing specifications are generally used to construct concrete protocols based on WS-CF. Any application that uses WS-CF must also decide what optional features are required. For the purpose of this document, the term *referencing specification* covers both formal specifications and more general applications that use WS-CF.

2 Introduction

Many protocols in distributed systems require software agents to perform a registration function to participate in the protocol. Examples of protocols that require explicit registration functions include notifications, transactions, virtual synchronous replica models based on group membership paradigms, and security. WS-Coordination Framework provides a WSDL interface for registering Web services as participants in arbitrary protocols. This is supported through the RegistrationService and Participant PortTypes.

Comment: Page: 1
Obviously I made these up and this would require consensus vote to adopt. However, I think we need to consider that the current interfaces will need to be renamed/reworked.

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 activity being performed, e.g., it may identify registration endpoints, 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. WS-Coordination Framework introduces a registration context type that builds on the context type defined in WS-Context to provide additional information required to enlist as a participant in an activity. Applications may use the registration context to define collections of services called “activity groups”. WS-Coordination Framework provides support for protocols that depend on group membership paradigms, such as coordination and security.

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 framework should provide enough flexibility and extensibility to its users that allow it to be tailored, statically or dynamically, to fit any requirement.

Comment: Need to settle on and consistently use a set of terms. For example, are problem domains the same as composite applications? Editorial issue.

This service builds upon WS-Context 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 that 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.

3 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.

3.1 Overview

WS-CF provides mechanisms for services to enlist with registration endpoints and 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 Context Service (WS-Context) specification and provides a registration context that leverages the activity model and context structure defined in WS-Context. In particular WS-CF:

- Allows services to register as participants in some protocol specific semantic;
- Introduces the notion of an activity group;
- Allows for the registration of participants in activity groups;
- Allows for propagation of group-specific information across the network by enhancing the default context structure provided by WS-Context;

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

- A Registration service: Provides an interface for the registration of participants within a specific protocol semantic.
- A Participant service, which may define the operation or operations that are performed as part of the protocol. It is possible to register participants that have no protocol specific callback operations.
- A registration context, which allows participants to join an activity group. The group membership facilities are used to build and manage relationships between services. For example, an activity group can be used as the basic definition of a participant set in a coordination protocol.
- A RecoveryCoordinator interface, which accommodates location migration for Participant service endpoints.

While the services that are described in WS-CF may be used without the facilities defined in WS-Context, the explicit encapsulation of group management concepts is based on extensions to the WS-Context specification. The activity model in WS-Context is extended to define the notion of an activity group. An activity group includes all services that register with for participation in a protocol in the context of an activity. The Registration service provides normative rules for registration in an activity group, thought he specification does not preclude the use of out-of-band or protocol specific enlistment mechanisms. This specification allows group membership to be managed with reference to a specific context; the relationship between different contexts is defined by the WS-Context specification; specific protocols based on activity groups may support subgroups and interposed activities. Activity groups are particularly useful for structuring relationships in the kinds of coordination protocols found in transaction systems and data replication/consistency protocols for clustered services.

Comment: Page: 1
This part really needs to be made longer – it really ought to provide an encapsulation of the abstract model. In particular it ought to lay out the concept of the activity group and the centrality of the group membership paradigm in detail.

Comment: We really need to determine what the relationship to context is, i.e., are group members (optionally) reflected in the context structure itself; when does membership in the activity group begin, when does it end – there should be some model level explanation, since the presence of group members in contexts opens up the possibility of autonomous entry into group membership via direct modification of the context, etc. Thoughts?

3.2 Invocation of Service Operations

How application services are invoked is outside the scope of this specification: they MAY use synchronous or asynchronous message passing.

Irrespective of how remote invocations occur, context information related to the sender's activity needs to be referenced or propagated. This specification determines the format of the context, how it is referenced, and how a context may be created.

In order to support both synchronous and asynchronous interactions, the components are described in terms of the behavior and the interactions that occur between them. All interactions are described in terms of correlated messages, which a referencing specification MAY abstract at a higher level into request/response pairs.

Faults and errors that may occur when a service is invoked are communicated back to other Web services in the activity via SOAP messages that are part of the standard protocol. The fault mechanism of the underlying SOAP-based transport isn't used. For example, if an operation fails because no activity is present when one is required, then it will be valid for the InvalidContextFault message to be received by the response service. To accommodate other errors or faults, all response service signatures have a generalFault operation.

Comment: Let's review this issue sometime in the TC in case there are some errors we should be using SOAP fault for.

Note: in the rest of this specification 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 shorthand 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 end-points are implemented and how they expose the various operations (e.g., via WSDL [1]) is not mandated by this specification. However, a normative WSDL binding is provided by default in this specification.

Comment: Whether WSDL is mandated or not seems related to the dependency on WSDL interfaces for defining operations, in any case I think it would be good to clear up the distinction between statements like WSDL isn't mandated yet there's a normative binding. Can we agree to mandate WSDL (and SOAP for that matter)?

> Note, this specification does not assume that a reliable message delivery mechanism has to be used for message interactions. As such, it MAY be implementation dependant as to what action is taken if a message is not delivered or no response is received.

3.3 Relationship to WSDL

Where WSDL is used in this specification it uses one-way messages with callbacks. This is the normative style. Other binding styles are possible (perhaps defined by referencing specifications), although they may have different acknowledgment styles and delivery mechanisms. It is beyond the scope of WS-Context to define these styles.

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 [1].

3.4 Referencing and addressing conventions

There are multiple mechanisms for addressing messages and referencing Web services currently proposed by the Web services community. This specification defers the rules for addressing SOAP messages to existing specifications; the addressing information is assumed to be placed in SOAP headers and respect the normative rules required by existing specifications.

Comment: Now that WS-Addressing is an open spec we should revisit the addressing format to see if it still makes sense to keep it generic or adapt WS-Addressing specifically.

However, the Context message set requires an interoperable mechanism for referencing Web Services. For example, context structures may reference the service that is used to manage the content of the context. To support this requirement, WS-CAF has adopted an open content model for service references as defined by the Web Services Reliable Messaging Technical Committee [5]. The schema is defined in [6][7] and is shown in Figure 1.

Deleted: Figure 1

Inserted: Figure 1

Deleted: Figure 1

```
<xsd:schema targetNamespace="http://docs.oasis-
open.org/wsrm/2004/06/reference-1.1.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="1.1">
  <xsd:complexType name="ServiceRefType">
    <xsd:sequence>
      <xsd:any namespace="##other" processContents="lax" />
```

```
    </xsd:sequence>
    <xsd:attribute name="reference-scheme" type="xsd:anyURI"
use="optional" />
  </xsd:complexType
```

Figure 1, service-ref Element

The ServiceRefType is extended by elements of the context structure as shown in Figure 2.

Deleted: Figure 2

Inserted: Figure 2

Deleted: Figure 2

```
<xsd:element name="context-manager" type="ref:ServiceRefType"/>
```

Figure 2, ServiceRefType example.

Within the ServiceRefType, the reference-scheme is the namespace URI for the referenced addressing specification. For example, the value for WSRef defined in the WS-MessageDelivery specification [4] would be http://www.w3.org/2004/04/ws-messagedelivery. The value for WSRef defined in the WS-Addressing specification [8] would be http://schemas.xmlsoap.org/ws/2004/08/addressing. The reference scheme is optional and need only be used if the namespace URI of the QName of the Web service reference cannot be used to unambiguously identify the addressing specification in which it is defined.

Messages sent to referenced services MUST use the addressing scheme defined by the specification indicated by the value of the reference-scheme element if present. Otherwise, the namespace URI associated with the Web service reference element MUST be used to determine the required addressing scheme. A service that requires a service reference element MUST use the mustUnderstand attribute for the SOAP header element within which it is enclosed and MUST return a mustUnderstand SOAP fault if the reference element isn't present and understood.

> Note, it is assumed that the addressing mechanism used by a given implementation supports a reply-to or sender field on each received message so that any required responses can be sent to a suitable response endpoint. This specification requires such support and does not define how responses are handled.

To preserve interoperability in deployments that contain multiple addressing schemes, there are no restrictions on a system, beyond those of the composite services themselves. However, it is RECOMMENDED where possible that composite applications confine themselves to the use of single addressing and reference model.

Because the prescriptive interaction pattern used by WS-Context is based on one-way messages with callbacks, it is possible that an endpoint may receive an unsolicited or unexpected message. The recipient is free to do whatever it wants with such messages.

4 WS-CF components

WS-CF provides four components that may be used to build collaborative protocols and complex composite applications: the Participant service, the Registration service, the Registration context, and Recovery service. The components are described in terms of their behavior and the interactions that occur between them. All interactions are described in terms of message exchanges, which an implementation may abstract at a higher level into request/response pairs or RPCs, for example. Like WS-Context, the components are organized in a hierarchical relationship, where individual components may be used without reference to higher level constructs that build on them. For example, the Registration and Participant services can be used without reference to an activity group.

Participant ServiceMany distributed protocols require software agents to enlist as participants within a protocol to achieve an application visible semantic. For example, participants may enlist in a transaction protocol in order to receive messages at coordination points defined by the protocol. 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 protocol model.

A Participant will use the message in a manner specific to the protocol and (optionally) return a result of it having done so. For example, upon receipt of a specific message, a Participant could commit any modifications to a database when it receives one type of message, or undo them if it receives another type. In some cases (ie, monitoring protocols) Participants may register for protocols that do not include any subsequent signaling. In other cases, such as publish-and-subscribe scenarios, Participants may register for a stream of messages that have no fixed semantic content with respect to the protocol itself. In general, rules governing the subsequent interaction between Participants and Registration endpoints are defined by specifications that make use of WS-CF.

In order to perform message exchanges with a Participant, two service roles are defined (illustrated in Figure 3), with the following operations (messages):

- The Participant: this accepts getStatus and AssertionType messages. The reply-to endpoint address is propagated with all of these messages. The callback messaging style and addressing rules for WS-CF are discussed in Section XXX.
- The ParticipantCallback: this accepts status, AssertionType, wrongState and generalFault call-back messages. Other error or fault messages are expected to be returned as specific instances of the AssertionType response.

The client sends an AssertionType message to the Participant with an accompanying reply-to address to a ParticipantCallback service 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 governing 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.

Comment: Given the vote to use specific participant WSDL in referencing specifications, does this not just go away?

Deleted: Figure 3

Inserted: Figure 3

Deleted: Figure 3

Comment: Page: 1
I guess this is an issue: what is the WSDL for this transmission? As it stands, I believe this should be struck, but I need some validation or correction here.

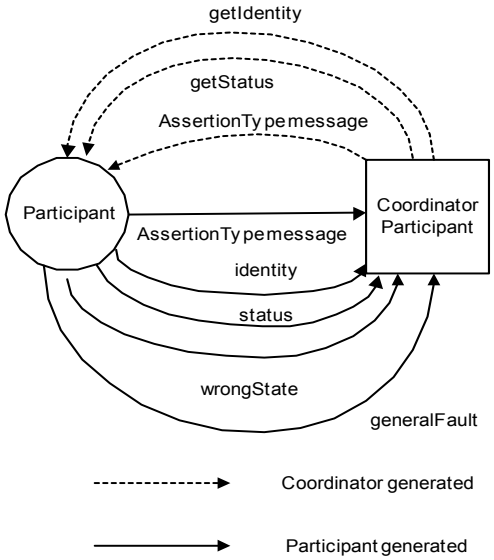

Figure 3, Coordinator-to-participant interactions.

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

Deleted: Figure 3

Inserted: Figure 3

Deleted: Figure 3

Deleted: Figure 4

Inserted: Figure 4

Deleted: 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

4.1 Registration Service

In order to become a Participant in a protocol, a service must first enlist with a Registration service. The protocol that the Registration implementation uses will depend upon the type of activity, application or service using the Registration service. For example, if the Registration 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 Registration service for a specific protocol(s) is located and ultimately registered with the Context Service is out of scope of this specification. An Registration service may identify the type of coordination protocol it supports deployment specific mechanisms may be used.

A Registration Service implementation provides:

- Support for the Participant service interface between Registration Service and Participant.

Operations on the Registration service MAY be implicitly associated with a Registration context, i.e., it is propagated to the Registration service in order to identify which activity group the Participant is interested in joining. Protocols that do not require explicit group membership support such as registration for publish-subscribe messages will invoke the Registration service without a Registration context. Protocols that rely explicitly on group membership like transactions or data replication will require that the Registration service be invoked with a Registration context.

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

4.1.1 Service-to-Registration interactions

These interactions define how a service may enlist or delist a Participant with the Registration service and perform other service-specific operations. The message exchanges are illustrated in Figure 5. They are factored into two different roles:

Deleted: Figure 5

Inserted: Figure 5

Deleted: Figure 5

- ServiceCoordinator: this accepts the addParticipant, removeParticipant, messages. All messages contain the ServiceRespondant endpoint for callback messages. The ServiceRespondant endpoint address is propagated on all of these messages.
- ServiceRespondant: this accepts the participantAdded, participantRemoved, qualifiers, generalFault, , wrongState, duplicateParticipant, invalidProtocol, invalidParticipant, participantNotFound messages.

addParticipant

This message is sent to the coordinator in order to register the specified Participant with the protocol supported by the Registration service. If the addParticipant message is contextualized with a RegistrationContext, the participant is added to the activity group identified in the context.

The protocol 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 protocols to enlist the participant, the a list of protocolType URIs may be propagated in the message. If any of the protocols are not supported by the Registration service then the invalidProtocol message will be sent to the ServiceRespondant indicating which protocol was at fault.

Comment: Do we want to say that registration is atomic (all or nothing) then?

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

Comment: Page: 1
First of all, these names are just way too confusing: participant coordinator, coordinator participant, etc. Second, we need to determine if this is even a supported interface. If so, I think it needs to be factored out to the highest level of the CF stack.

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 Registration service more than once and the protocol does not allow this, then the duplicateParticipant message is sent to the ServiceRespondant

removeParticipant

This message causes the Regisration service to delist the specified Participant . If the Participant is associated with an activity group, it is removed from the activity group. If the Participant has not previously been registered with the Registration service for the specified protocol, then it will send the participantNotFound message to the ServiceRespondant.

Removal of a participant need not be supported by the specific protocol implementation and may also be dependant upon where in the protocol the system is as to whether it will allow the participant to be removed.The rules governing removal of participants from participtionin a protocol or activity group are governed by referencing specifications. For activity groups, if the Activity has begun completion, or has completed, then the wrongState message is sent.

Comment: There seems to be some confusion overall in the definition of the protocol as to which part belongs in the referencing spec and which part belongs in the WS-CF spec (i.e. is there a minimal protocol or set of operations that each protocol must support, such as getStatus).

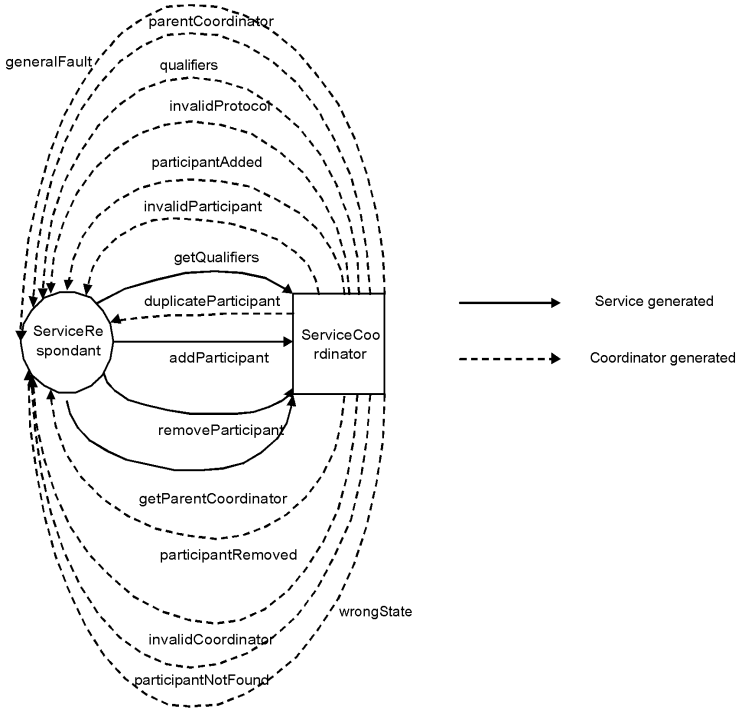

Figure 5, Service-to-coordinator interactions.

Comment: Page: 1
Requires modification

The ServiceRespondant and ServiceCoordinator roles are elucidated in WSDL form in Figure 6.

Deleted: Figure 6

Inserted: Figure 6

Deleted: Figure 6

```
<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 6, WSDL portType Declarations for ServiceRespondant and ServiceCoordinator Roles.

4.1.2 Client-to-coordinator interactions

These interactions (illustrated in Figure 7) 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.

Comment: Page: 1
Is this an out of band problem? At a minimum, coordination should be layered on top of the rest of this stuff, so this should be moved to the end of this section.

Deleted: Figure 7

Deleted: Figure 7

Inserted: Figure 7

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 Context Service, or may be specific to the coordination protocol, identified by its QName, is returned to the ClientRespondant via the status message.

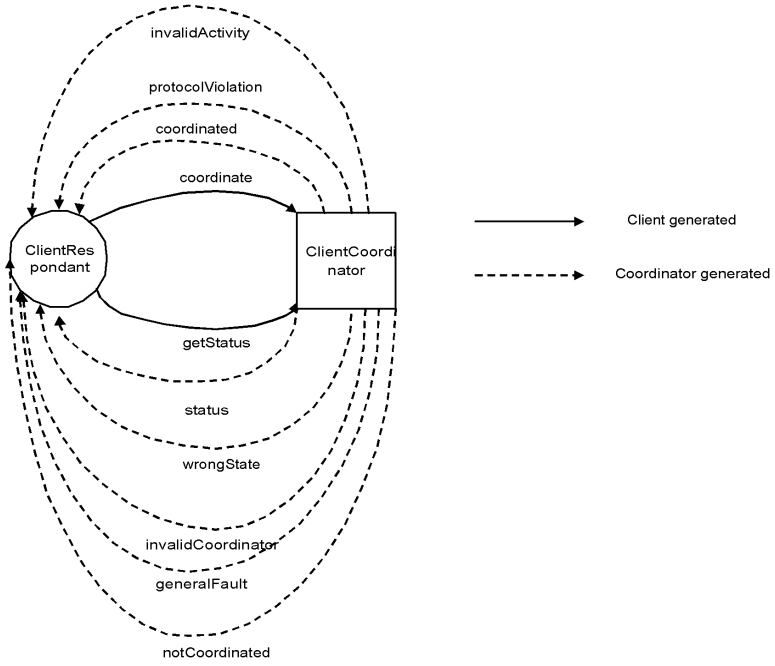

Figure 7, Client-to-coordinator interactions.

The ClientRespondant and ClientCoordinator roles are shown in WSDL form in Figure 8.

Deleted: Figure 8

Inserted: Figure 8

Deleted: Figure 8

```
<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 8, WSDL portType Declarations for ClientRespondant and ClientCoordinator Role

4.1.3 Registration Context

In order to support registration in activity groups. Activity groups are described in Section XXX., it is necessary for the participants to be enlisted in the activity group via some mechanism. This specification defines a Registration service to support enlistment in n activity group.. In a distributed environment, this requires information about the Registration service (essentially its network endpoint) to be available to remote participants. The Context 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 and optional information associated with the demarcation activity and management of the context. However, it has been designed to be extensible such that additional, service-specific information may be added to the contextby other specifications.

Comment: Need to include this description somewhere.

```
<xs:complexType name="ContextType">
  <xs:complexContent>
    <xs:extension base="wsctx:ContextType">
      <xs:sequence>

      <xs:element name="registration-service"
type="wsctx:ServiceRefType"
        maxOccurs="unbounded"/>
      <xs:any namespace="##any" processContents="lax"
maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Comment: Page: 1
This needs a lot of work to bring it up to speed with WS-Context. I would favor renaming this explicitly to RegistrationContextType. Moreover, I strongly believe that we ought to reexamine the idea of a list of participating services in this layer. It didn't have a clear enough semantic (in my view at least) in WS-Context, so we struck it. In WS-CF, I believe we might have an appropriate home for the idea.

Comment: Page: 1
Need to check the type for consisteny with how this is used in WS-CTX.

Figure 9, WS-CF ContextType derives from the WS-Context ContextType.

The Registration context contains the following elements in addition to the WS-Context ContextType structure:

- A service reference to a Registration service. This enables Participant services to be enlisted or delisted in an activity group.
- XXXparticipant list? (see comment)

Comment: Page: 1
Should we, as a second bullet item, consider formally defining the list of participants in an activity group? This would be useful for peer discovery of participants, particularly when network access to the Registration service is unavailable.

Comment: Are we dropping this since it was dropped from WS-Context?

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

```
<context
xmlns="http://www.webservicestransactions.org/schemas/wsctx/2003/03"
    timeout="100">
    <context-identifier>
      http://www.webservicestransactions.org/wsctx/abcdef:012345
    </context-identifier>
    <activity-service>
      http://www.webservicestransactions.org/wsctx/service
    </activity-service>
    <type>
      http://www.webservicestransactions.org/wsctx/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/wsctx/5e4f2218b
        </context-identifier>
       <activity-service>
         http://www.webservicestransactions.org/wsctx/service
       </activity-service>

<type>http://www.webservicestransactions.org/wsctx/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>
```

Comment: Page: 1
This needs to be cleaned up a bit.

Interposition

Comment: Editorial issue – might be better to move this toward the end of the spec.

Consider the situation depicted in Figure 10, where there is a registration service and three participants. If we assume that each of these participants are on the same machine, but which is different to the registration service, then there is obviously an overhead involved in registering each of them separately.

Deleted: Figure 10
Inserted: Figure 10
Deleted: Figure 13

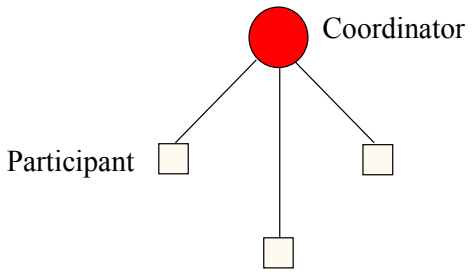

Figure 10. Coordinator-participant distributed interactions.

Deleted: *10*
Deleted: *13*
Inserted: *10*

A common approach to reducing this overhead is to allow a participant on each machine to act as a proxy registration service. This technique of using proxies (or subordinates) is known as interposition. Each domain that imports a context may create a subordinate service that enrolls with the imported registration service as though it were a participant. Interposition obviously requires the importing domain to use a different context when communicating with services and participants within that domain since the registration service endpoint will be different, as shown in Figure 11.

Deleted: Figure 11
Inserted: Figure 11
Deleted: Figure 14

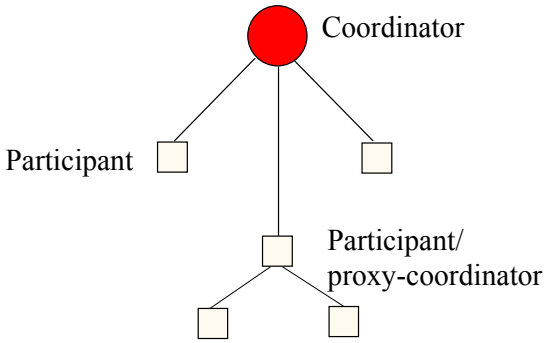

Figure 11. Participant coordinator.

Deleted: *11*
Inserted: *11*
Deleted: *14*

4.2 Recovery Service

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

Comment: Another editorial issue – draw a clearer distinction between what happens at the coordinator/protocol level and what happens within the execution environment? I.e. when do signals pass between one and the other?

- Unlike in a traditional transactional system, where crash recovery mechanisms are only responsible for guaranteeing consistency of object data, applications that use Registration

Service's will typically also require the ability to recover the participants and potentially the activity group structure that was present at the time of the failure, enabling the application to progress onwards.

The following roles are defined to assist in recovery; the message interactions are shown in Figure 12:

Deleted: Figure 12

Inserted: Figure 12

Deleted: Figure 10

- 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 callback results.
- RecoveryParticipant: this service is used to return the recovery information to a recovering participant via callbacks. It supports the recovered, status, wrongState and generalFault messages.

recover

This operation is used by participants that have previously successfully enlisted with a Registration service. 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 Registration service. The recover operation allows the participant to inform the Registration service that the participant has moved from the original address to a new address. It may also be used to start recovery operations by the protocol engine.

If successful, the recoverResponse message is sent to the RecoveryParticipant. If the recovery handshake occurs in the context of an activity, the message also contains the current status of the activity. This status may be used by the recovering participant to perform local recovery operations, although this will depend upon the 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.

Comment: Page: 1
I had deleted this above, but at this point this would have to be unknownActivity, no?

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

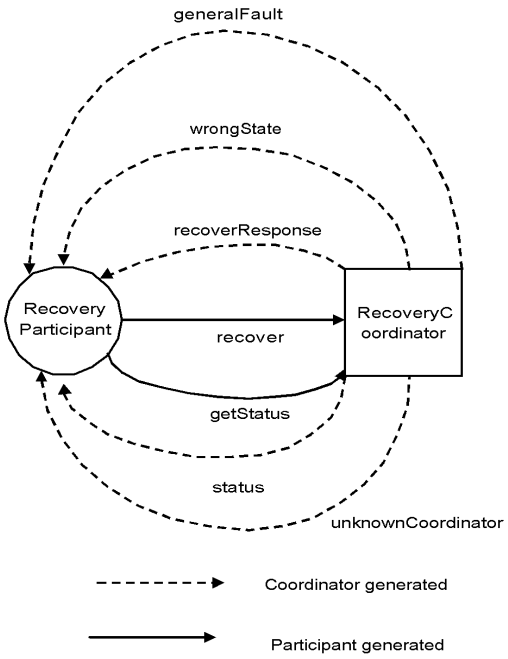

Figure 12, Participant recovery.

The RecoveryCoordinator and RecoveryParticipant interfaces are presented in Figure 13.

```
<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 13, WSDL portType Declarations for RecoveryParticipant and RecoveryCoordinator Roles

Deleted: *12*
Deleted: *10*
Inserted: *12*
Deleted: Figure 13
Inserted: Figure 13
Deleted: Figure 11
Deleted: *13*
Inserted: *13*
Deleted: *11*

5 References

Comment: May need additional references for WS-Context, WS-TXM, and WS-Addressing

[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