Web Services Coordination Framework Specification (WS-CF)

Editors draft version 0.3

26 March 2005

Abstract

WS-CF defines interfaces that drive the coordination of multiple Web service executions related in an activity, according to the requirements of a WS-TXM protocol type such as ACID, long running actions, or business process, or of a protocol type defined in another specification.

WS-CF defines an open, pluggable coordination framework that supports multiple protocol types. The coordination framework ensures the set of Web service participants in an activity is notified of actions required of them, and that any protocol actions initiated by the participants are communicated to the other participants, to ensure a common outcome.

Coordination in general refers to the ability of multiple Web services to act in combination through a software agent such as a broker, even though they were not designed to do so, and conform to a common, predefined outcome such as commit, rollback, or compensate, based upon conditions recognized and acted upon by the protocol.

Coordination is required in a variety of different aspects of distributed applications, such as orchestration, workflow, atomic transactions, caching and replication, security, auctioning, and business-to-business activities.

The fundamental capability offered by the WS-CF specification is the ability to register a web service as a participant in an activity.

WS-CF extends the WS-Context late binding session model SOAP messages processed within the scope of an activity contain context headers that uniquely identify a single activity. WS-CF extends the session model using a registration context. Registration in the context of an activity adds the registered service to an activity group. Membership in the group drives a 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.

Table of contents

Deleted: 5
Deleted: 5
Deleted: 5
Deleted: 5
Deleted: 6
Deleted: 9
Deleted: 11
Deleted: 11
Deleted: 11
Deleted: 12
Deleted: 12
Deleted: 13
Deleted: 13
Deleted: 13
Deleted: 14
Deleted: 16
Deleted: 22
Deleted: 23

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/2005/02/wscf
```

1.1.1 Prefix Namespace

Prefix	Namespace
Wscf	http://docs.oasis-open.org/wscaf/2005/02/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

1.2 Referencing Specifications

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

A referencing specification generally defines the protocol types based on WS-CF. Any protocol type that uses WS-CF must specify what optional features are required.

WS-CF uses WS-CTX as a referenced specification, and WS-TXM uses WS-CF as a referenced specification.

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. The WS-Coordination Framework provides a WSDL interface for registering Web services as participants in various protocols types, as defined using referencing specifications.

Context information in support of a registration action 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 context 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-CF 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.

2.1 Definitions

- Protocol type: A set of messages exchanged among participants in an activity for the purpose of determining or executing a common outcome agreed upon by all participants.
- Coordination: The act of a software agent exchanging messages with the participants in an activity for the purpose of determining a common outcome.
- Composite application: An application comprised of multiple Web services (including their execution or implementation environments) joined to achieve a common purpose.
- Common outcome: A way in which Web services in a composite application can agree in common as to whether or not the desired purpose of the composite was achieved.
- Activity: See also WS-Context. An activity represents a mechanism external to WS-CF according to which multiple Web services are placed in combination to achieve a common goal.
- Registration: The act of an individual Web service within a composite application of registering to participate in a given protocol type.
- Termination: The end or completion of a given protocol type so that the participants in an activity can agree upon a common outcome, as defined by the protocol type.
- Activity group: (Do we need a separate definition for an activity group?)

A Web service becomes a participant in an activity through its inclusion in an orchestration flow or other means by which Web services can be combined into a composite application. An activity becomes known to a coordinator via the registration of the individual Web services within the activity for inclusion within a particular protocol. Various protocol types can be used to drive a common outcome among the services, such as two-phase commit, compensations, and asynchronous business process management. When a Web service registers, it registers for a particular protocol type. The set of Web services in an activity group therefore is defined as the set of services registering on behalf of the activity for the same protocol type.

The coordination protocol is executed using a sequence of correlated one-way message exchange patterns. The use of correlated one-ways is required because HTTP is an unreliable

transport, and a coordinated protocol type needs to know whether or not a message was received and processed.

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 an interface for services to enlist with a coordinator for a specific protocol type, 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 WS-Context specification to provide 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 a protocol;
- Introduces the notion of an activity group;
- Allows for the registration of participants in activity groups;
- Allows for propagation of group-specific protocol information 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, which provides an interface for the registration of participants within a specific protocol.
- A participant service, which defines the operation or operations that are performed as part of the protocol.
- A registration context, which allows participants to join an activity group.

The group membership facilities are used to build and manage relationships among services. For example, an activity group can be used as the basic definition of a participant set for a given coordination protocol.

WS-CF builds upon the activity concept defined in the WS-Context specification by narrowing the notion of an activity to that of an *activity group*: such a group contains members (participants) that will be driven through the same protocol. WS-CF says nothing about specifics of such coordination protocols and when or where participants may join and leave: this is left up to the protocol types.

Because WS-CF is meant to support a range of coordination protocols, each possessing different protocol messages and potentially different coordinator interfaces, WS-CF does not define how or when coordination occurs. This is left to the protocol types.

WS-CF defines the activity group and associated service (the *Registration Service*). The group paradigm is central to coordination, whether it is coordinating the outcome of distributed transactions, security domains, replica consistency, cache coherency etc. The activity group is tied to an underlying WS-Context activity such that their lifetimes coincide.

Web services that wish to join or leave the group use of the Registration Service. The membership of the group may also be obtained from the Registration Service. Specific implementations of the Registration Service may impose restrictions on how and when group membership changes may occur; these are outside the scope of the WS-CF specification. In addition, some uses of group membership may place constraints on consistent views of group membership, particularly in the presence of member failures.

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.

3.2 Invocation of Service Operations

How application services are invoked is outside the scope of this specification, however context information related to the sender's activity needs to be referenced and/or propagated.

All interactions are described in terms of correlated messages, which a referencing specification MAY abstract at a higher level into request/response pairs. 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.

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.

The WSDL binding is normative; however other implementations that are semantically equivalent and preserve interoperability are allowed.

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. If an operation fails because no activity is present when one is required, then the InvalidContextFault message will be sent to the requester. To accommodate other errors or faults, all response service signatures have a generalFault operation and as a transientFault operation.

> Note, a transientFault message is produced when the implementation finds it cannot successfully execute the requested operation at that time from some *temporary* reason. This reason may be implementation or referencing specification specific. A receiver of a transientFault is free to retry the operation which originally generated it on the assumption that eventually a different response will be produced. Sub-types of transientFault MAY be further defined using the fault model described which can allow for the communication of more specific information on the type of fault.

Comment: We should add a fault section when we final agree on the overall fault model to use, and this note should go there.

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 may be used as long as interoperability is preserved, although they may have different acknowledgment styles and delivery mechanisms. It is beyond the scope of WS-Coordination Framework to define these styles.

> Note, conformant implementations MUST support the normative WSDL defined in the specification where those respective interfaces are required. WSDL for optional components in the specification is REQUIRED only in the cases where the respective components are supported.

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.

However, the Coordination Framework 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.

Comment: [I think we need to acknowledge and use WS-Addressing as the primary and single addressing format – WS-Addressing should be a normative spec - I realize this will have to be brought up at the TC however – and depending on the outcome of the discussion this section may have to be rewritten – Eric]

```
<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.

```
<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-Coordination Framework 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 five components that may be used to build collaborative protocols and complex composite applications: the Participant service, the Registration service, and the Registration context. 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.

4.1 Participant Service

Many 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 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 (e.g., 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. As such, there is no defined WSDL interface defined for the Participant Service; it is an abstract entity that is given concrete representation by referencing specifications and is only discussed within the scope of this specification for clarity of the overall model concept.

4.2 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 within an extended transaction infrastructure, then one protocol type 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 a Registration service for a specific protocol(s) is located and ultimately registered with the Context Service is out of scope of this specification. A Registration service MAY identify the type of coordination protocol it supports using deployment specific mechanisms.

A Registration Service implementation provides support for Registering Services to enlist Participant Services with a specific activity group. 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 the specific activity group.

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

4.2.1 Service-to-Registration interactions

These interactions define how a service (the *Registering Service*) may enlist or delist a Participant (Service) with the Registration Service. The message exchanges are illustrated in Figure 3. They are factored into two different roles:

Registration Service: this accepts the addParticipant, removeParticipant, recoverParticipant, registrationRecovered and getStatus messages. All messages contain the Registering Service endpoint for callback messages, although it is OPTIONAL as to whether the Registration Service remembers these beyond a specific interaction.

Registering Service: this accepts the participantAdded, participantRemoved, participantRecovered, status, recoverRegistration, generalFault, wrongState, duplicateParticipant, invalidProtocol, invalidParticipant, and 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. A valid RegistrationContext MUST accompany this message and the participant will be added to the activity group identified in the context. This context MAY be passed by reference or by value. It is implementation dependant as to whether any context information other than the basic reference values is required.

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 list of protocolType URIs may be propagated in the message. The Registration Service MUST ensure that all protocols specified are supported before any registration happened. If some of the protocols are not supported by the Registration service then no registration occurs and the invalidProtocol message MUST be sent to the Registering Service indicating which protocols were at fault.

Upon success, the Registration service calls back to the Registering Service with the participantAdded message, including in this message a unique OPTIONAL endpoint reference that MAY be used by the Registering Service or Participant Service for further interactions. How and when this endpoint reference should be used is outside the scope of this specification and is left to referencing specifications to determine. For example, it may be used by a coordination service to refer to the endpoint that the participant should use for the coordination protocol.

A referencing specification MAY decide to send the wrongState message if the Activity has begun completion, or has already completed when this operation is attempted.

The termination of the activity group MAY be triggered by the completion of the WS-Context service activity.

If the same participant has been enrolled with the Registration service more than once and the referencing specification does not allow this, then the duplicateParticipant message is sent to the ServiceRespondant. How the registration of the same participant multiple times is dealt with at the protocol level is outside the scope of this specification and is left to referencing specifications to define, as the rules governing the protocol are defined by a referencing specification

removeParticipant

This message causes the Registration service to delist the specified Participant. A valid RegistrationContext MUST accompany this message to identify the activity group from which the participant should be removed. This context MAY be passed by reference or by value. It is implementation dependant as to whether any context information other than the basic reference values is required. If successful, the ParticipantRemoved message is sent to the invoker.

If the Participant has not previously been registered with the Registration service for the specified activity group, then it will send the participantNotFound message to the Registering Service.

Removal of a participant need not be supported by the specific protocol and may also be dependant upon where in the protocol the system is as to whether a referencing specification will allow the participant to be removed. The rules governing removal of participants from participation in a protocol or activity group are governed by referencing specifications. A referencing specification MAY decide to send the wrongState message if removal is disallowed; for example, the Activity has begun completion, or has already completed when this operation is attempted.

In addition, some protocols may allow for Registration service to autonomously delist Participant services. In this case, the Registration Service will send an unsolicited ParticipantRemoved message to the service that was responsible for enlisting the Participant.

recoverParticipant

Comment: replaceParticipant ?

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

A valid RegistrationContext MUST accompany this message in order to identify the group in which the failed participant previously existed. This context MAY be passed by reference or by value. It is implementation dependant as to whether any context information other than the basic reference values is required.

If successful, the participantRecovered message is sent to the invoker. 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 recovery 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 invalidActivityFault 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 Registering Service by the coordinator.

If the Registration Service cannot deal with recovery of the participant for a temporary reason, the transientFault message is sent and the receiver MAY try again.

recoverRegistration

Comment: replaceCoordinato r?

This operation on the Registering Service MAY be used by a recovered Registration Service to indicate that it has recovered on a new endpoint address. When a Registration Service fails and subsequently recovers it may not be able to recover at the same address that prior Registering Services used to enlist with the Registration service. This OPTIONAL operation allows the Registration Service to inform Registering Services that it has moved from the original address to a new address. It may also be used to start recovery operations by the protocol engine.

The use of recoverRegistration SHOULD only be attempted when the Registration Service has failed and recovered on another endpoint because to do otherwise MAY result in continued use of stale RegistrationContext information elsewhere in the application; the context refers to the old endpoint address for the Registration Service.

A valid RegistrationContext MUST accompany this message. This context MAY be passed by reference or by value. It is implementation dependant as to whether any context information other than the basic reference values is required.

If successful, the registrationRecovered message is sent to the Registration Service. 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 recipients to perform local recovery operations, although this will depend upon the protocol in use

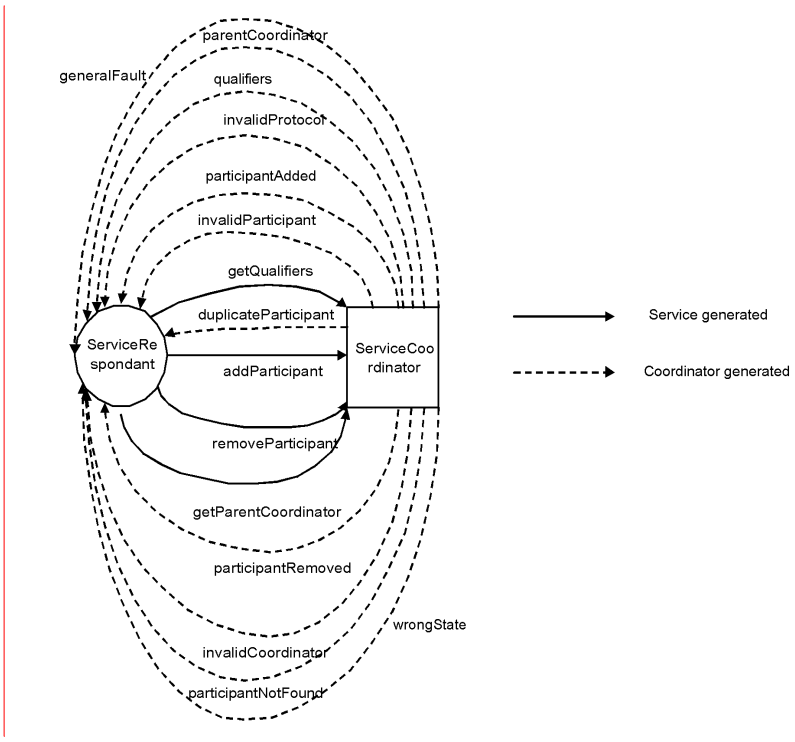
If the Registering Service cannot be located, then the unknownService message is sent back.

If the Registering Service cannot deal with recovery of the Registration Service for a temporary reason, the transientFault message is sent and the receiver MAY try again.

getStatus

The status of the activity group may be obtained by sending the getStatus message to the recovery coordinator. A valid RegistrationContext MUST accompany this message. This context MAY be passed by reference or by value. It is implementation dependant as to whether any context information other than the basic reference values is required.

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 invoker via the status message. GetStatus will return the same Status value that is returned by the getStatus operation on the Context Service, assuming the queries occur at the same point in the activity lifecycle.



Comment: To be updated.

Figure 3, Service-to-coordinator interactions.

The Registration Service and Registering Service roles are elucidated in WSDL form in Figure 4.

Comment: All of this needs changing.

```
<wsdl:portType name="RegistrationServicePortType">
  <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="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="parentCoordinator">
    <wsdl:input message="tns:ParentCoordinatorMessage"/>
  </wsdl:operation>
  <wsdl:operation name="generalFault">
    <wsdl:input message="tns:GeneralFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="invalidActivity">
    <wsdl:input message="wsctx:InvalidActivityFaultMessage"/>
  </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>
```

Comment: Asw?

Figure 4, WSDL portType Declarations for Registration Service and Registering Service Roles.

4.2.2 Registration Context

In order to support registration in activity groups 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 an activity group. In a distributed environment, this requires information about the Registration service (essentially its network endpoint) to be available to remote participants. The WS-Context provides mechanisms for propagating basic context information betweenservices. 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. WS-Coordination Framework extends the ContextType defined in WS-Context to allow services to register as Participants in an activity. The RegsitrationContextType is shown in Figure 5.

```
<xs:complexType name="RegistrationContextType">
  <xs:complexContent>
    <xs:extension base="wsctx:ContextType">
      <xs:sequence>
      <xs:element name="registration-service" type="ref:ServiceRefType"
        minOccurs="1"/>
      <xs:any namespace="##any" processContents="lax" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
```

```
</xs:complexType>
```

Comment: This seems broken: no subprotocols and no list of participants!

Figure 5, WS-CF RegistrationContextType 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)

The XML below shows an example of a Registration context for 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: 16
This needs to be cleaned up a bit. Is ANYTHING correct here at this point?

Comment: Change.

4.3 Interposition

Consider the situation depicted in Figure 6, 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

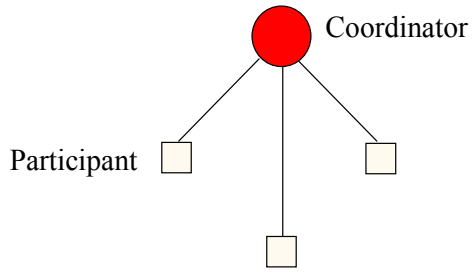


Figure 6, Coordinator-participant distributed interactions.

Deleted: *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 7,

Deleted: Figure 11

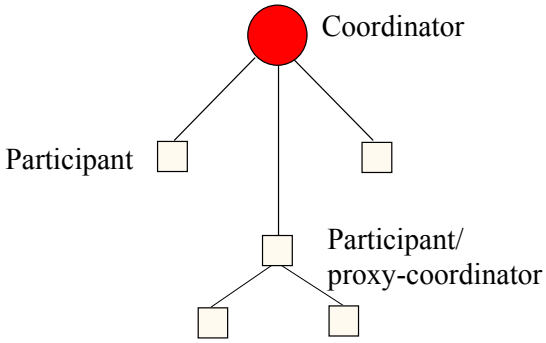


Figure 7, Participant coordinator.

Deleted: *11*

5 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

[3] OASIS Web Services Context Specification,

[4]

Comment: What should this reference?