Web Services Coordination Framework Specification (WS-CF)

~~Committee~~Editors draft version ~~0.2~~0.3

~~22 June 2004~~

4 April 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 ~~a requirement present~~required in a variety of different aspects of distributed ~~applications. For instance,~~applications, such as orchestration, 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~~ activities.

~~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 Context Service.~~

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

1 Note on terminology .... 6~~4~~

1.1 Namespace 6

1.1.1 Prefix Namespace 6

1.2 Referencing Specifications........ 6

2 Introduction 7

3 WS-CF architecture........ 9~~7~~

3.1 Overview 9

3.2 Invocation of Service Operations .... 13~~8~~

3.3 Relationship to WSDL........ 14~~8~~

3.4 Referencing and addressing conventions........ 14~~8~~

4 WS-CF components .... 16~~10~~

4.1 Participant Service........ 17~~10~~

4.2 Registration Service........ 21~~11~~

4.2.1 Service-to-Registration interactions........ 21~~11~~

addParticipant .... 22~~11~~

removeParticipant .... 22~~12~~

recoverParticipant .... 24~~12~~

recoverRegistration .... 25~~13~~

getStatus........ 25~~13~~

4.2.2 Registration Context .... 27~~15~~

4.3 Interposition........ **Error! Bookmark not defined.**~~16~~

5 References........ 39~~17~~

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 a referencing 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

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-CTXContext 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.

2.1Problem statement

Define a specification for a generic coordination service for a Web Services, to be known as the WS-CF, utilizing the Web Services CTXContext 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 CTXContext Service.
- Define the mappings onto the Web Service environment (SOAP message and header definitions, context definition, endpoint address requirements, etc.).
- Define the required infrastructure to support such as an event mechanisms, etc.
- Define the roles and responsibilities of WS-CF subcomponents (e.g., Coordination Service Participants).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 ~~Extended coordination models~~Overview

~~The WS-CF~~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 ~~Web Services CTXContext Service (WS-CTXContext) specification and provides a coordination service that plugs into WS-CTXContext.~~defined in WS-Context. 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;~~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;
- ~~Propagates coordination-specific information across the network~~Allows for propagation of group-specific protocol information by enhancing the default context structure provided by ~~WS-CTXContext;~~WS-Context;
- The main components involved in using and defining the WS-CF are:

~~- 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.~~

~~- A Participant: The operation or operations that are performed as part of coordination sequence processing.~~

~~- A Coordination Service: Defines the behaviourbehavior 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-CTXContext 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-CTXContext. Web services are shown as circles. The mandated WS-CF services are the CoordinationServiceALS and the CTXContext Service, whereas the optional services which may be provided through non-WS-CF routes are the Application Web Service, Coordination Service and Participant.

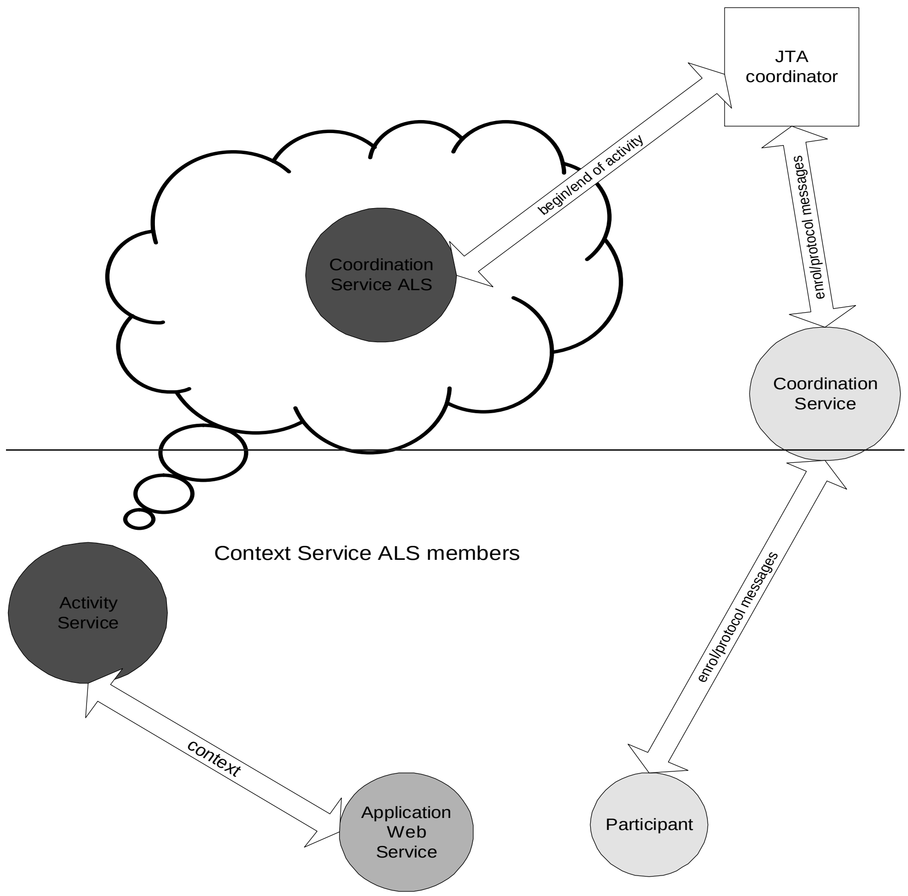


Figure 1, WS-CF services.

3.2 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.

3.3 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.

4Coordination and activities

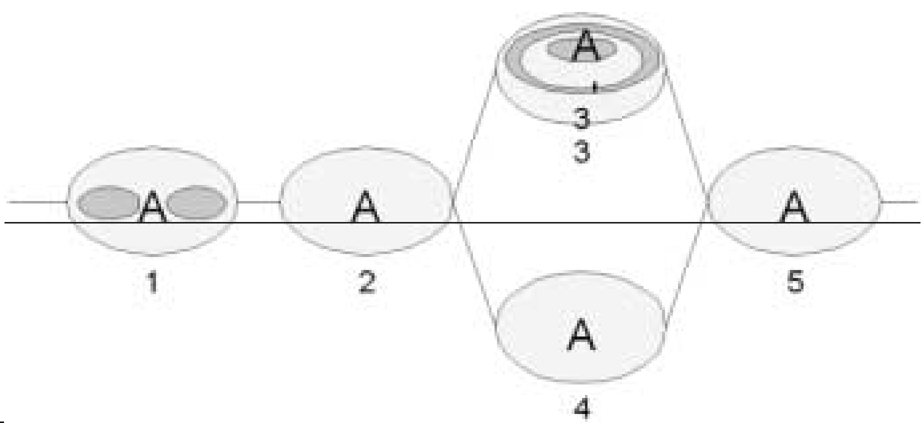
In the WS-CTXContext specification it was shown how the framework manages the lifecycle of Activities, which are used to scope application and service specific work, along with the associated Activity contexts necessary for distributed invocations. It also described how services can be plugged into this 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.

4.1Activity 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.~~

~~4.2Coordination 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.~~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.

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 3.

```
<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 3, service-ref Element

The ServiceRefType is extended by elements of the context structure as shown in Figure 4.

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

Figure 4, 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 behaviou~~rbehavio~~r and the interactions that occur between them. All interactions are described in terms of message ~~messages,~~exchanges, 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~~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 ~~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.~~

~~5.1Participants~~can be used without reference to an activity group.

4.1 Interposition

WS-CF supports the notion of *interposition*: where a Participant Service that is enlisted with a Registration Service also behaves as a Registration Service to other Participant Services. In this way, WS-CF supports the building of graphs and trees by the addition of participants to an activity structure that are themselves registration endpoints.

The technique of interposition uses proxies (or subordinates). Each domain that imports a WS-CF context MAY create a subordinate registration service that enrolls with the imported registration service as though it were a participant. This specification does not prescribe how and when this may occur. Interposition then requires the importing domain to use a different context when communicating with services and participants that are required to register with the subordinate registration service, as shown in Figure 5.

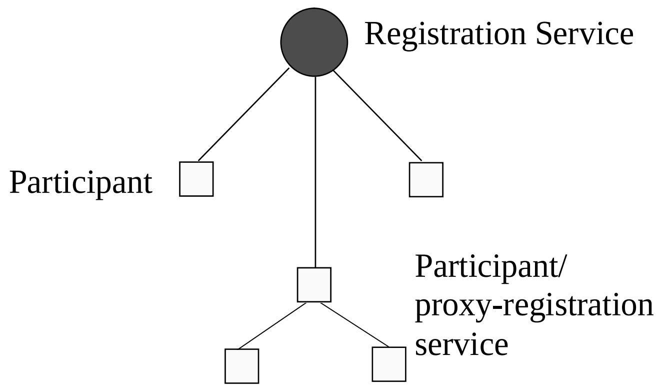


Figure 5, Participant coordinator.

This specification does not define what are allowable forms of graphs that may be created using interposition. Such definitions are the responsibility of referencing specifications.

4.2 Participant Service

~~At~~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 ~~application or service, messages are communicated between a coordinator and registered participants through the exchange of protocol specific messages. For example, 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 coordination~~protocol model.

A Participant~~(coordination participant)~~ will use the message in a manner specific to the ~~Coordination Service and~~protocol and (optionally) 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.

In some cases (e.g., monitoring protocols) ~~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.~~

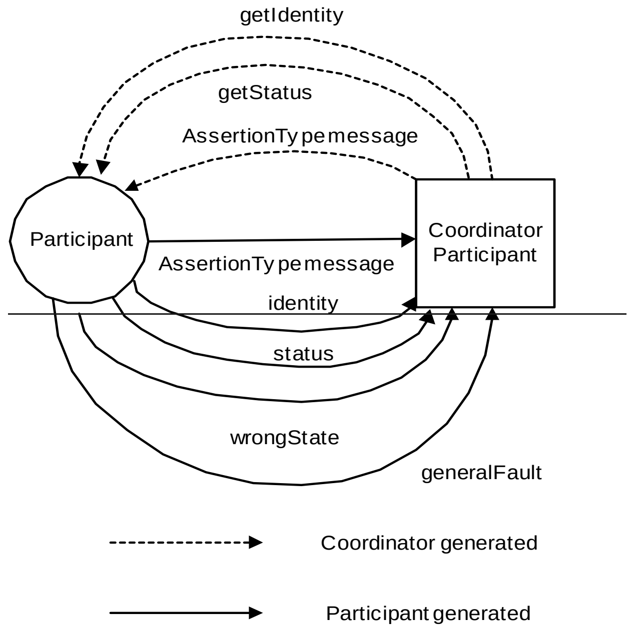
~~In order to perform the necessary interactions for coordinator-to-participant, two service roles are defined (illustrated in Figure 3), with the following operations (messages):~~

~~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):~~

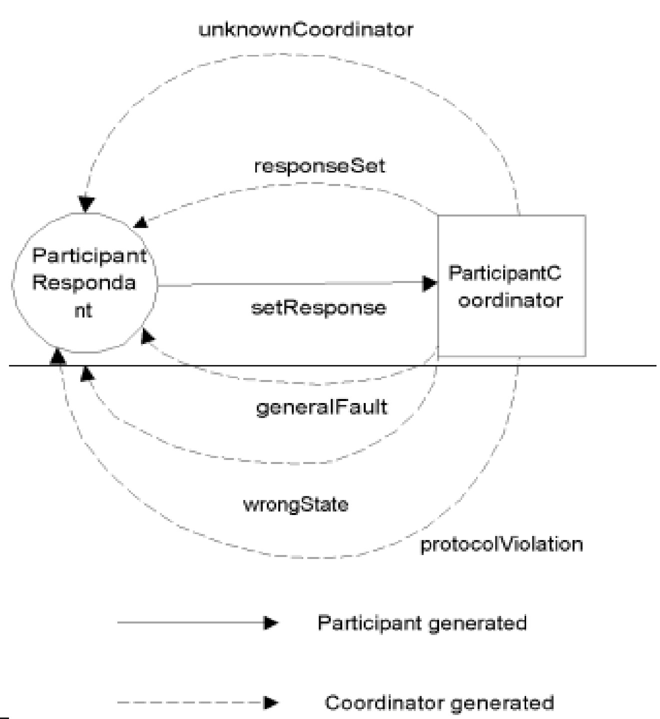
~~ParticipantCoordinator: this accepts the setResponse message. The endpoint address for the ParticipantCoordinator is returned to the Participant during the registration process (see below). The ParticipantRespondant address is propagated on all of these messages for call-back response messages.~~

~~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.

5.2Qualifiers

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

5.3Coordinator

An activity coordinator is associated with each activity; this happens implicitly through the appropriate Activity Lifecycle Service (ALS) that is enlisted with the CTXContext 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.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.

4.3 Registration Service

~~The protocol that the coordinator~~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 ~~coordination~~Registration service. For example, if the ~~coordination~~Registration service is being used~~for~~ within an extended transaction infrastructure, then one protocol ~~implementation~~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 ~~an ALS~~a Registration service for a specific~~coordination~~ protocol(s) is located and ultimately registered with the C~~TXC~~ontext Service is out of scope of this specification. ~~An ALS may~~A Registration service MAY identify the type of coordination protocol it supports ~~via the ALS identify message, but other~~using 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~~Registration 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 CTXContext Service and Participant.~~

~~All operations on the coordinator service are~~provides support for Registering Services to enlist Participant Services with a specific activity group. Operations on the Registration service MAY be implicitly associated with ~~the current~~a Registration context, i.e., it is propagated to the ~~coordinator~~Registration service in order to identify ~~which coordinator is to be operated on.~~the specific activity group.

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

~~5.3.1~~4.3.1 ~~Service-to-coordinator~~Service-to-Registration interactions

These interactions define how a service (the *Registering Service*) may enlist or delist a ~~participant with the coordinator and perform other service-specific operations, and~~Participant (Service) with the Registration Service. The message exchanges are illustrated in Figure 11~~Figure 8~~. They are factored into two different roles:

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

~~endpoint address is propagated on all of these messages.~~

~~- ServiceRespondant:~~Registering Service: this accepts the participantAdded, participantRemoved, ~~qualifiers, parentCoordinator,~~participantRecovered, status, recoverRegistration, generalFault,~~unknownCoordinator,~~ wrongState, duplicateParticipant, invalidProtocol, invalidParticipant, and participantNotFound messages.

addParticipant

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

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

~~ServiceRespondant.~~

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

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.

~~If~~A referencing specification MAY decide to send the wrongState message if the Activity has begun completion, or has already ~~completed, then the wrongState message is sent.~~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 ~~coordinator~~Registration service more than once and the ~~coordination protocol~~referencing specification does not allow this, then the duplicateParticipant message is sent to the ~~ServiceRespondant.~~

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 ~~If the participant is invalid within the scope of the coordinator, the invalidParticipant message is sent to the ServiceRespondant.~~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 ~~coordinator to remove the specified Participant from the ActivityCoordinator identifier in the associated context.~~ 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 ~~coordinator~~Registration service for the specified ~~coordination protocol,~~activity group, then it will send the participantNotFound message to the ~~ServiceRespondant.~~Registering Service.

~~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~~protocol and may also be dependant upon where in the protocol the ~~coordinator~~system is as to whether ~~it~~a referencing specification will allow the participant to be ~~removed.~~

removed. The rules governing removal of participants from participation in a protocol or activity group are governed by referencing specifications. ~~If~~A referencing specification MAY decide to send the wrongState message if removal is disallowed; for example, the Activity has begun completion, or has ~~completed, then the wrongState message is sent.~~already completed when this operation is attempted.

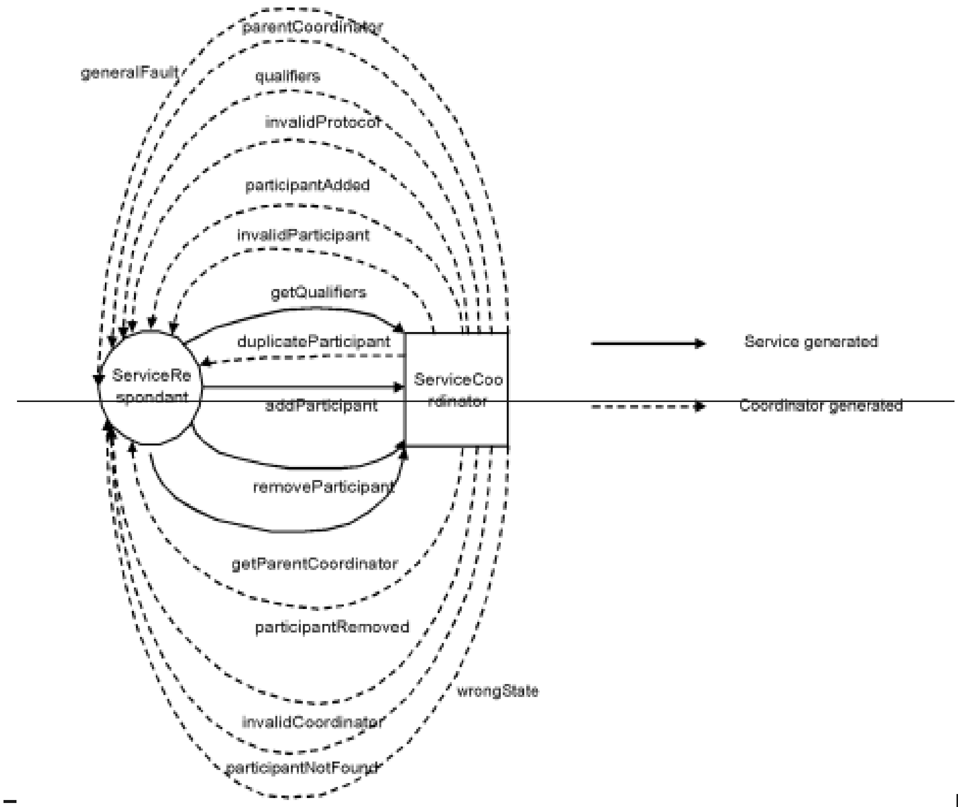
~~getParentCoordinator~~

~~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.~~



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

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

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.

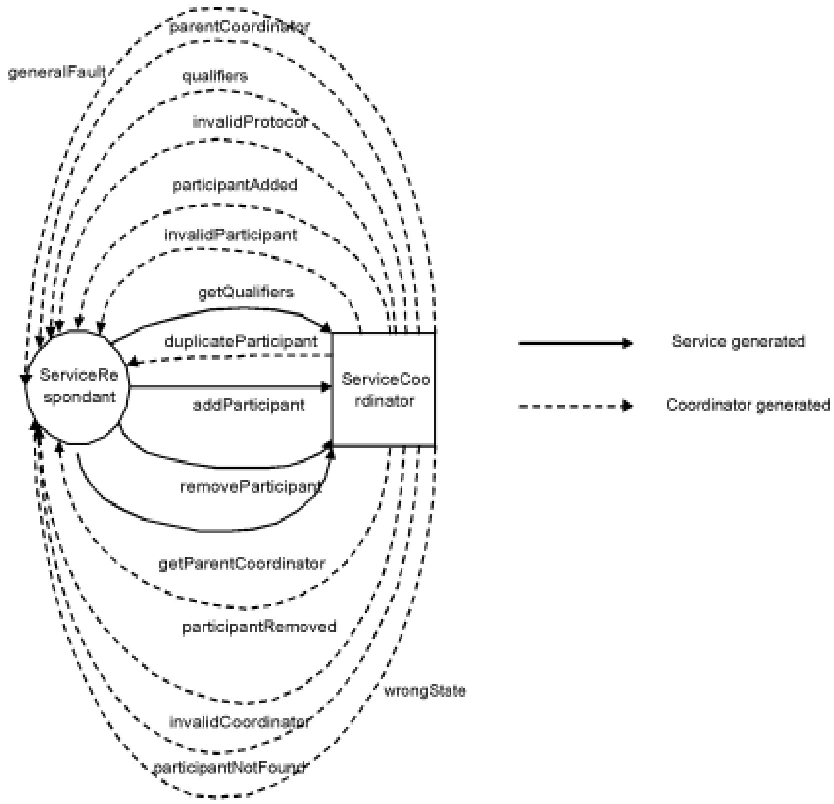


Figure 11, Service-to-coordinator interactions.

The ~~ServiceRespondant and ServiceCoordinator~~Registration Service and Registering Service roles are elucidated in WSDL form in Figure 1213~~Figure~~ Figure 1213~~Figure 9.~~.

```
<wsdl:portType
name="ServiceCoordinatorPortType">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="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">name="invalidActivity">
      <wsdl:input
message="tns:UnknownCoordinatorFaultMessage"/>message="wsctx:InvalidActi
vityFaultMessage"/>
    </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 12~~13~~9~~, WSDL portType Declarations for ServiceRespondant and ServiceCoordinator~~, WSDL portType Declarations for Registration Service and Registering Service Roles.

~~5.3.2Client-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 CTXContext Service, or may be specific to the coordination protocol, identified by its QName, is returned to the ClientRespondant via the status message.

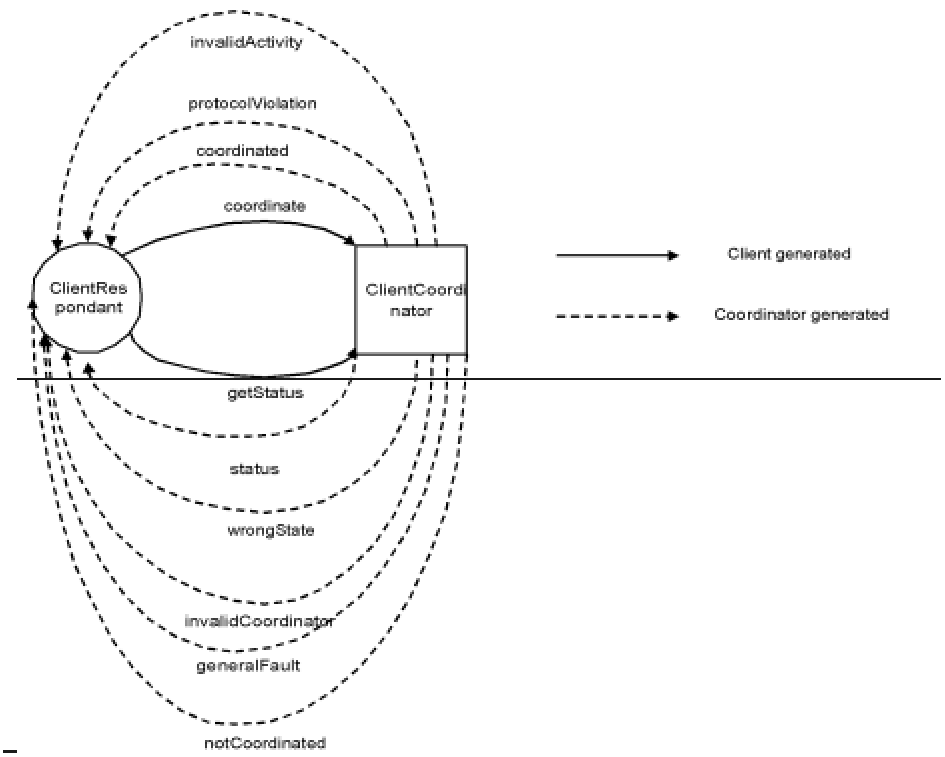


Figure 10, Client-to-coordinator interactions.

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*~~

~~5.3.3~~4.3.2 ~~Context enhancement~~Registration Context

In order to ~~perform coordination,~~support registration in activity groups it is necessary for the participants to be ~~enrolled with coordinators.~~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 ~~coordinator~~Registration service (essentially its network endpoint) to be available to remote participants. The ~~CTXContext Service is already responsible~~WS-Context provides mechanisms for propagating basic context information~~between distributed activities.~~betweenservices. 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~~ identity and optional information associated with the demarcation activity and management of the context. WS-~~hierarchy of coordinator references.~~

~~<xs:complexType name="ContextType">~~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="protocol-reference" 
type="tns:ProtocolReferenceType"/>
      <xs:element name="coordinator-reference" 
type="tns:CoordinatorReferenceType" 
        maxOccurs="unbounded"/>name="registration-service" 
type="ref:ServiceRefType" 
        minOccurs="1"/>
      <xs:any namespace="##any" processContents="lax" 
maxOccurs="unbounded"/>minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 161~~712, WS-CF ContextType~~, WS-CF RegistrationContextType derives from the WS-C~~TXC~~ontext 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 ~~coordination~~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>
```

~~5.4Interposition~~

~~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.~~

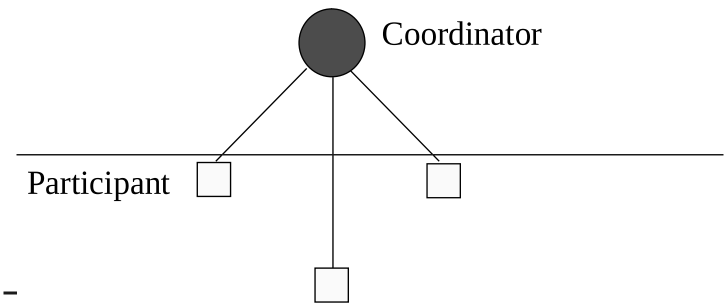


Figure 13, Coordinator-participant distributed interactions.

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.

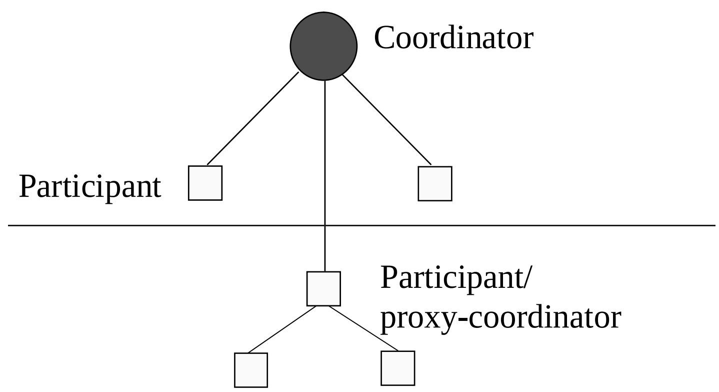
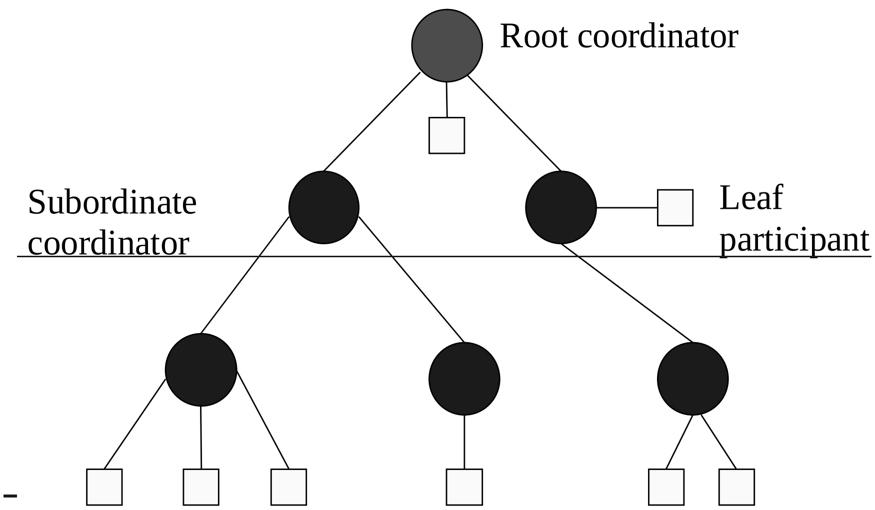


Figure 14, Participant coordinator.

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.~~

~~5.5State 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 CTXContext Service, or may be specific to the coordination protocol, identified by its QName, is returned to the RecoveryParticipant via the status message.

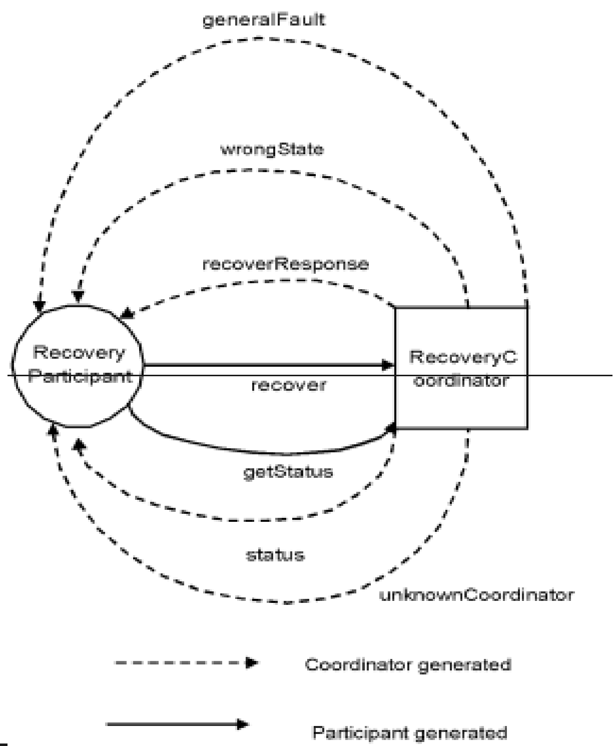


Figure 16, Participant recovery.

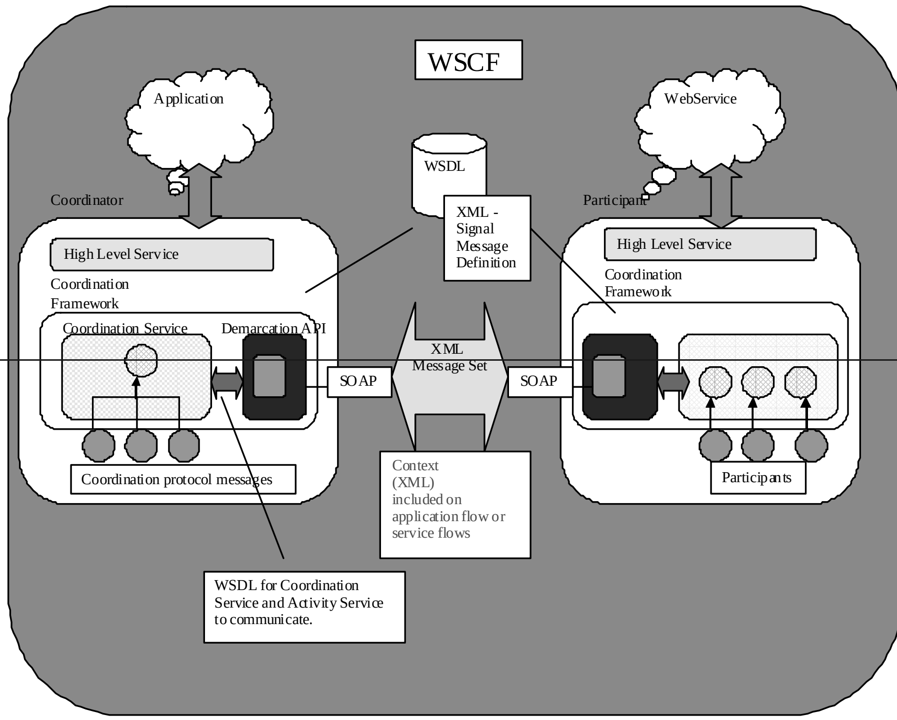
The RecoveryCoordinator and RecoveryParticipant interfaces are presented in Figure 17.

```
<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

6Roles & 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.

6.1Coordination Service Activity Lifecycle Service provider

This Web service ties into the WS-CTXContext 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.

6.2Coordination 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.

6.3Web 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 CTXContext 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.

7Example

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 Figure 19is 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.

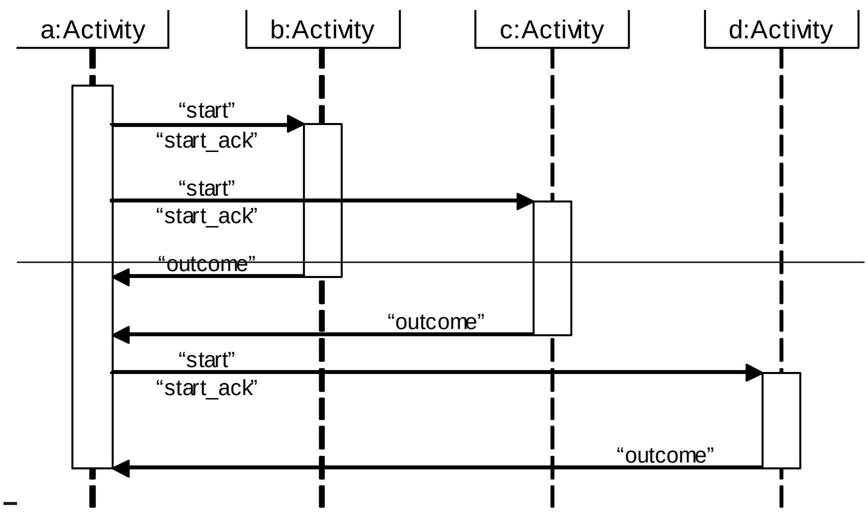


Figure 19, Workflow coordination.

8 Issues

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

- 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.
- 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.
- 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.
- 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.

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~~http://www.w3.org/TR/wsdl

[3] OASIS Web Services Context Specification,

[4]