Web Services Context (WS-Context) Ver1.0

July 28, 2003

Authors:

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

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL THE OWNERS OR THEIR LICENSORS BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION, LOST REVENUE, PROFITS OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO ANY FURNISHING, PRACTICING, MODIFYING OR ANY USE OF THE SPECIFICATION, EVEN IF THE OWNERS AND/OR LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You will indemnify, hold harmless, and defend the Owners and their licensors from any claims based on your use of the Specification for any purposes other than those of internal evaluation, and from any claims that later versions or releases of any Specifications furnished to you are incompatible with the Specification provided to you under this license.

Restricted Rights Legend

If this Specification is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in the Specification and accompanying documentation shall be only as set forth in this license; this is in accordance with 48 C.F.R. 227.7201 through 227.7202-4 (for Department of Defense (DoD) acquisitions) and with 48 C.F.R. 2.101 and 12.212 (for the non-DoD acquisitions).

Report

You may wish to report any ambiguities, inconsistencies or inaccuracies you may find in connection with your evaluation of the Specification ("Feedback"). To the extent that you provide the Owners with any Feedback, you hereby: (i) agree that such Feedback is provided on a non-proprietary and non-confidential basis, and (ii) grant the Owners a perpetual, non-exclusive, worldwide, fully paid-up, irrevocable license, with the right to sublicense through multiple levels of sublicensees, to incorporate, disclose, and use without limitation the Feedback for any purpose related to the Specification and future versions, implementations, and test suites thereof.

Abstract

The ability to scope arbitrary units of work is a requirement in a variety of aspects of distributed applications such as workflow and business-to-business interactions. By scoping work, we mean that it is possible for business activity participants to be able to determine unambiguously whether or not they are participating in the same activity.

In order to correlate the work of participants within the same activity, it is necessary to propagate additional information know as the context to each participant. The context contains information (such as a unique identifier) that allows a series of operations to share a common outcome. For example, in a Web services-based application, a SOAP header block might contain contextual information that is propagated when invoking an operation on a Web Service, or when multiple participants exchange SOAP messages in order to create a larger interaction such as a process flow.

A Web Services Context Service maintains a repository of context information and tracks contexts shared between multiple participants in Web services interactions. An Context Service can also be a participant within an activity, creating a tree to further propagate the context. A Web Services Context Service accepts and emits SOAP messages for interoperability with any type of participant, regardless of operating system, programming language, or platform – and is independent of underlying transfer or transport protocols.

Context is always propagated in addition to application payload, where context information travels within the SOAP header blocks while application payload (that is, the content intended for processing by a SOAP node playing the ultimateReceiver role) is propagated inside the SOAP body.

The context information is specific to the type of activity being performed, such as to identify a transaction coordinator, the other participants in an activity, or recovery information in the event of a failure, etc. Therefore, a single context type is not sufficient for all types of activity that a Context Service may be required to support. Hence, the capabilities of the Context Service must be extensible in an application specific manner and services must be able to augment the context as they require to suit their own particular application domains.

Given the importance of context propagation in many distributed systems, including Web Services, standardization on a context framework (Context Service) is a logical progression in increasing the usefulness and robustness of the Web Services architecture. That is, there is a distinct requirement for a generic context propagation service that allows users and services to register with it, and customize it on a per-service or per-application basis. This service also supports newly emerging Web Service standards such as coordination, workflow and transactions.

Status of this document

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

Table of contents

1. Introduction

The Web Services Context Service (WS-CTX) is the first of three parts of the Web Services Composite Application Framework (WS-CAF) specification. These specifications are intended to be useable in isolation; we group them here only for the purposes of showing how they can be employed in a particular critical use case. The other two parts are:

- Web Services Coordination Framework (WS-CF)
- Web Services Transaction Management (WS-TXM)

The WS-CTX specification defines the scope of context sharing for other services and specifications to use; WS-CAF uses this as the basic unit of WS-CAF processing. WS-CTX is intended as a lightweight mechanism for allowing multiple Web services to share a common context. Web services share a common context when they perform related activities such as performing multiple interactions with a data management resource, interactive display, or automated business process.

WS-CTX defines the context, the scope of context sharing, and basic rules for context management. WS-CTX describes how to define an *Activity* in which multiple Web services are related through shared context. The responsibility for actions in relationship to the context resides with the individual Web services within the Activity.

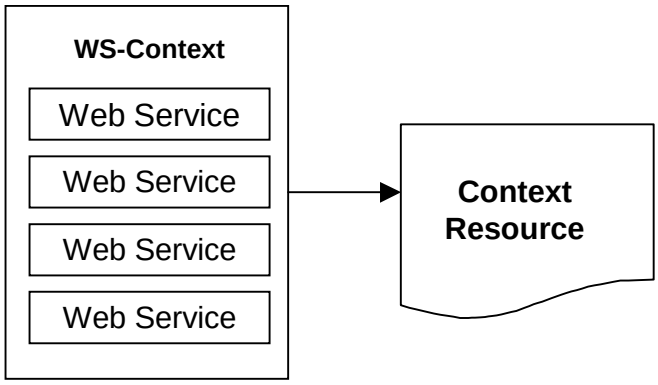

Figure 1, Web Services CTX Concepts

Figure 1 illustrates the basic concepts within the Web Services-Context. A group of Web services is related to each other through their association with a Web Services Context Service that manages a shared context for the group.

2. Scope of Sharing

Web services are increasingly being used as an integration platform for distributed enterprise applications. The resulting applications can be very complex in structure, with complex relationships between their constituent applications, compounded by the need to share common data or context across multiple interactions. Furthermore, the execution of such an application may take a long time to complete, and may contain

long periods of inactivity, often due to the constituent applications requiring user interactions.

The ability to scope arbitrary units of distributed work is a requirement in a variety of aspects of distributed applications, e.g., quick updates, workflow, business-to-business interactions, automated business processes, and others. By scoping work, it is possible for an activity's participants to be able to determine unambiguously whether or not they are in the same activity. This is the fundamental functionality of a Web services Context Service, which may be implemented using a separate, distinct, service or by adding semantics to the behaviour of participants.

In order to correlate the work of distributed participants within the same activity, it is necessary to propagate additional information (the *context*) to participants. The context contains information such as a unique ID that allows a series of operations to share a common outcome. For example, a SOAP header block might contain context information that is propagated when interacting with a service, or when multiple participants exchange SOAP messages in order to create a larger interaction such as a process flow or other aggregation of Web services.

The activity context information is represented as a Web resource, accessible by dereferencing its URI. Context operations may flow implicitly (transparently to the application) with normal messages sent to participants/endpoints during Web service execution, or may be an explicit action on behalf of a participant. Context information flows within SOAP header blocks with normal payloads sent to participants/endpoints. A single context type is not sufficient for all uses of an Context Service. Hence, it must be extensible in an application specific manner: services must be able to augment the context as they require. Furthermore, it may be required that additional application specific context information flow to these participants or the services which use them.

Context propagation is a fundamental requirement of many distributed systems, including Web services. However, the type of context information that is used may vary depending upon the circumstances, e.g., in an atomic transaction system is may be a URI for the coordinator, whereas for secure data interchange it may we the sender's public encryption key. Therefore, what is needed is standardization on a context framework (*Context Service*) and a minimum definition of context that allows services to register with it, and customize it on a per activity basis. The Context Service is intended to be flexible and extensible to support newly emerging Web services standards such as workflow and transactions. This specification presents a suitable Context Service architecture and protocol set.

Problem statement

Define a specification for a generic context propagation service for Web Services, to be known as the Web Services Context Service (WS-CTX). Outline the necessary infrastructure and protocol requirements to support such a service.

The WS-CTX as proposed can be used in conjunction with other business protocol activities, e.g., coordination and choreography. The WS-CTX also includes a

description of the protocol, infrastructure and required environmental definitions needed to control the activities associated with an application.

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

The goals of the specification are to:

- Define the notion of an activity for Web Services and the core infrastructure necessary to support this (e.g., a *context service*).
- Provide a basic definition of a core infrastructure service consisting of an Context Service for the Web Services environment, i.e., the WS-CTX.
- Define the mappings onto the Web Services environment (SOAP message and header definitions, context definition, endpoint address requirements, etc.).
- Define the required infrastructure support such as event mechanisms, etc.
- Illustrate common usage patterns (e.g., sharing common resources, activity coordination, transactions etc.).

3. WS-CTX architecture

The section outlines the architecture of WS-CTX, describing the components that implementations provide and those that are required from applications. The WS-CTX represents a Web services interaction of a number of *activities* related to an overall application. In particular it:

- Defines demarcation points which specify the start and end points of an activity.
- Registers Web services so that they that can become participants in the lifecycle of an activity and manage and augment the context associated with that activity. The participants are notified of the lifecycle of an activity (e.g., when it starts or ends) and contacted when context information is to be propagated, and when the overall results of the activity require further action by the individual participants.
- Propagates context information (essentially information about the activity structure at the sender) across the network, either by reference (its URI) or by value (the "contents" of the URI).

WS-CTX is not aimed specifically at a single service type or application domain: it is a more low-level and fundamental service concerned purely with the management of abstract activity entities through shared context. The main components of the WS-CTX are:

1) A *Context Service*: defines the scope of an activity and how information about it (the *context*) can be referenced and propagated in a distributed environment. Activities can be hierarchically structured, such that nesting and concurrent activities are possible. The Context Service may be a Web Service that is physically remote from, or collocated with its users.

2) A *context*: defines basic information about the activity structure that is identified using a URI, propagated on application messages. The context contains information necessary for multiple Web services to be associated with the same activity. This information may be dynamically augmented by applications and services. Activities are managed by the Context Service, which maintains a repository of shared contexts associated with execution environments; whenever messages are exchanged within the scope of an activity, the Context Service can supply the associated context which may then be propagated with those messages.

3) *The Activity Lifecycle service*: these Web services are registered with the core Context Service and are informed of the lifetime of an activity and may further enhance it by suitable higher-level interfaces. For example, a security service implementation may define an activity to represent the scope of a secure interaction between business participants and may then provide explicit authentication interfaces on top of the Context Service. Whenever a context is required for the activity associated with the current execution environment, the Context Service calls each registered ALS and obtains from it an addition to the basic context; from this it eventually assembles the entire context data that can be propagated.

The core Context Service is concerned with managing the lifecycle of context propagation.

How application services are invoked is outside the scope of this specification: they may use synchronous RPC-style approaches or asynchronous message passing.

Irrespective of how remote invocations occur, context information related to the sender's activity hierarchy will need to be referenced or propagated and this specification determines how the format of the context, how it is referenced, and how that context is created.

Services and applications can customize and add to this framework in whatever manner they require and can enroll with it to enable them to control various aspects of the lifecycle of an activity. By default, when an activity begins and ends all that the core framework does is ensure that the context is initialized and the activity execution sequence is managed appropriately with respect to the context.

An activity is initiated by an initial Web service request. The initiating Web service is informed when an activity begins and terminates and has access to the shared context. For example, if a coordination service initiates an activity, it will be informed when an activity terminates and can control that termination in a domain specific manner, e.g., using a two-phase or other completion protocol.

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 [1]. Complete WSDL is available at the end of the specification.

4. Relationships of Activities and contexts

An increasing number of distributed applications are constructed through the composition of existing applications and services. The resulting applications can be very complex in structure, with complex relationships between their constituent applications. Furthermore, the execution of such an application may take a long time to complete, and may contain long periods of inactivity, often due to the constituent applications requiring user interactions, i.e., these applications require control to flow from process to process, and for this flow to be orchestrated.

An *activity* is a unit of (distributed) work, involving one or more parties (services, components, objects). An activity is *created*, *runs*, and then *completes*. An *outcome* is the result of a completed activity, which may be used to determine subsequent flow of control to other activities, and may result in failure notification messages to the participants of an activity. Activities may execute over long periods of time (minutes, hours, days, …). As we shall see, during its lifetime an activity may require periods of coordination between its constituent parties (participants) or may need to interact with other services such as transaction management and security.

An activity is essentially a way of *scoping application specific work*. The definition of precisely what an activity is and what services it will require in order to perform that work, will depend upon the environment and application in which it is used. We split the services that are involved within an activity into the following classifications:

- Activity Lifecycle Service (ALS) – these services participate in the life-cycle of the activity (are informed when it starts, ends etc.) and may augment it for their own purposes. For example, a security ALS may define the scope of an activity to be the scope of a security domain. Multiple ALS-es may be enrolled with the activity.
- Application Services (AS) – these are the services that the application uses to obtain the functional aspects of the task. For example, a stock-purchase application may determine that it has several activities which involve determining current stock quantities, placing orders for new stock, and finally delivery of that new stock. Alternatively, an on-line book provider may divide its activities into users placing books into a shopping basket, the user then confirming the books that are to be purchased and providing credit card details, etc.

In both cases the definition of the activity is directly related to the need for a group of Web services to share common information, such as a stock price or a book inventory quantity. An activity is itself implemented as a Web Service, and a Web Service may be enrolled with an activity in order to define it.

The relationship between ALS and Context Service, Application Services and Applications is shown in Figure 2.

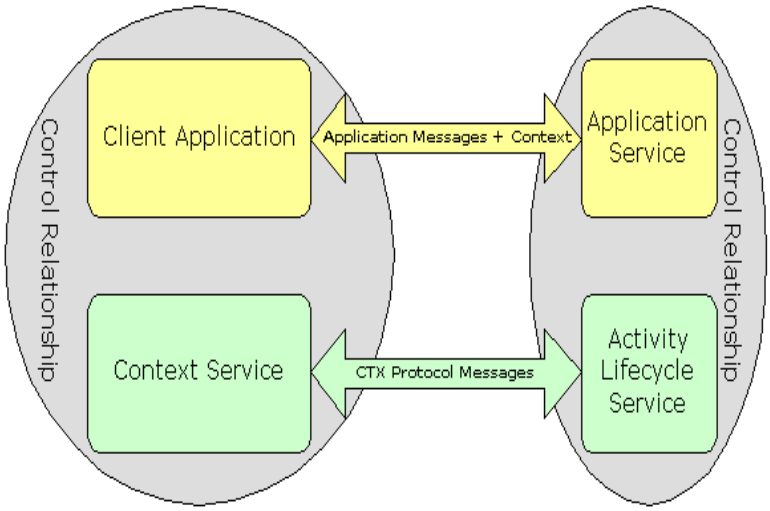

Figure 2, Context Service Component Relationships

Context Service and the Web Services stack

The relationship between the Context Service and the current Web Services stack is shown in Figure 3. It is expected that the services above the Context Service will be able to fully utilize the facilities it provides.

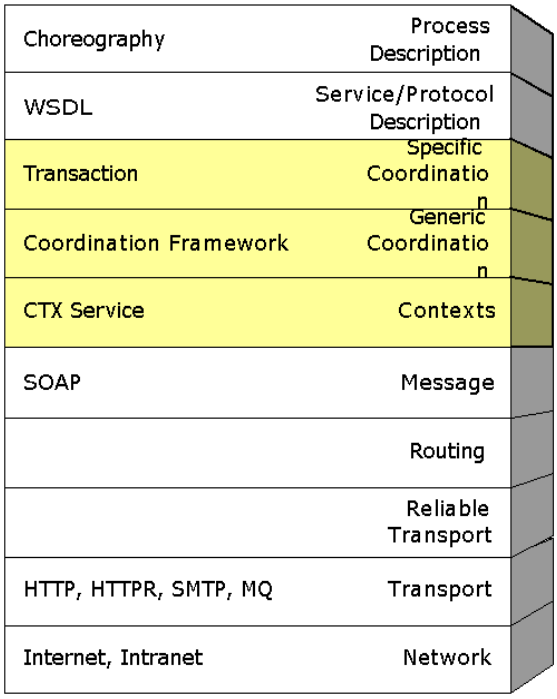

Figure 3, Context Service and the Web Services stack.

The Context Service framework

The first component of the WS-CTX is an *activity framework* (the *Context Service*), that supports the abstract notion of an activity and allows applications and services to scope work within these activities by sharing context. The basic infrastructure simply

supports the lifecycle of activities and ensures that each is uniquely identified. In addition, information about activities can be propagated between execution environments.

For example, an event management system may wish to be informed when an activity starts and ends so that it can disseminate this information to interested parties. Likewise, a transaction service may need to be informed if the activities represent transactions, so that it can ensure any transaction context information is updated appropriately. The basic Context Service simply manages activity lifecycles and their associated contexts; for example, the start of a new activity may implicitly cause the service to add an entry to a context hierarchy and the termination of the activity may result in an entry being removed from the hierarchy.

Composite activities

Activities can be nested to arbitrary degree. An activity, which contains component activities, *may* impose a requirement on an ALS implementation for managing these component activities. However, the core Context Service framework does not specify how (or even if) activities should be coordinated: that is the domain of a coordination service ALS which may be enrolled with the framework.

Contexts

In order for an activity to span a number of Web services, certain information has to flow between the sites/domains involved in the application. This information is commonly referred to as the *context* and includes information related to the services associated with the framework (e.g., security). It is referenced or propagated in a distributed environment to provide continuity of context information between remote execution environments, and may be transported for example inside SOAP header blocks. This may occur transparently to the client and application services or may be part of an explicit exchange.

As we have described, the Context Service framework is simply a means to scope distributed work by associating it with a unique identifier and share related context. The core context propagation framework must provide a generic context structure that enables an activity to be uniquely identifiable such that work can be correlated. Furthermore, it must provide support for applications and services to extend the context structure in an implementation specific manner, e.g., for coordination, transactions, security or replication. This context is shown in Figure 4.

```
<xs:complexType name="ContextType">
  <xs:sequence>
    <xs:element name="context-identifier" type="xs:anyURI"/>
    <xs:element name="activity-service" type="xs:anyURI" minOccurs="0"/>
    <xs:element name="type" type="xs:anyURI" minOccurs="0"/>
    <xs:element name="activity-list" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="service" type="xs:anyURI" minOccurs="0"
            maxOccurs="unbounded"/>
```

```
        </xs:sequence>
        <xs:attribute name="mustUnderstand" type="xs:boolean" use="optional"
          default="false"/>
        <xs:attribute name="mustPropagate" type="xs:boolean" use="optional"
          default="false"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="child-contexts" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="child-context" type="tns:ContextType"
            maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="timeout" type="xs:int" use="optional"/>
</xs:complexType>
```

Figure 4, Context Service Context

The context consists of a the following items:

- A mandatory URI identifier called context-identifier. This guarantees global uniqueness for an individual activity (such an identifier can also be thought of as a “correlation” identifier or a value that is used to indicate that a task is part of the same work activity.
- An optional element, activity-service, which identifies the Context Service responsible for generating the context;
- An optional identifier that indicates the type of the activity;
- An optional list of the services currently participating in the activity, called participating-services;
- An optional list of child activities, in the child-contexts element;
- A timeout value, which indicates how long the context information is valid for; after this period has elapses, the activity structure it refers to may no longer exist (e.g., the activity may be terminated and the structure garbage collected). This is to allow an Context Service implementation, in the event of no external stimulus, to terminate activities automatically rather than have them potentially run forever.

Context propagation may also occur using different protocols than those used by the application. The Context Service does not assume a specific means by which contexts are propagated, leaving this up to the implementation.

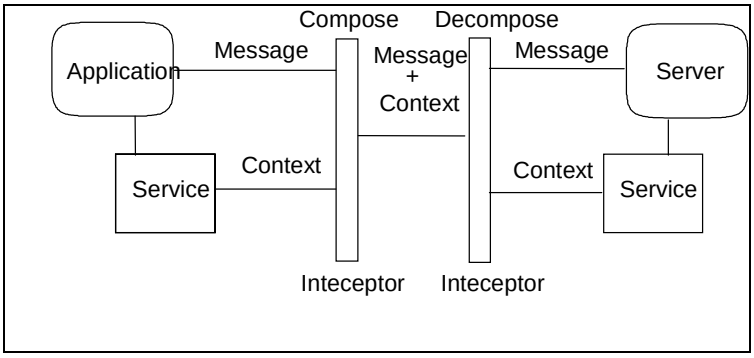

Figure 5, Services and context flow.

The Context Service maintains context information on behalf of every activity. This information may be propagated in-line with application invocations or may simply be referenced by its URI. The context may be built up in two ways:

- Service invocations: as each service is invoked it may augment or otherwise manipulate the context that it receives and propagates.
- ALS: because each ALS participates in the lifecycle of an activity, the Context Service may build up an initial context based on these service invocations.

As we have already described, WS-CTX considers the context to be a first-class entity. As such, an application may choose to propagate either the entire contents of the context with messages or simply the URI of the context, which represents the service (component of the Context Service) that maintains the context information. It is important to note that in those situations where only the context-identifier element is present in a context, that the URI which it contains can be de-referenced to obtain the full context (to use or update its information), which includes the full set of activity contexts.

The choice of whether to transmit a full or abbreviated context is left to the user. It is however expected that when dealing with large context elements (those which subsume a large number of activities) that the URI-only form will be used for efficiency.

Figure 6, shows the message interactions for the context using the call-back style mentioned earlier: solid lines represent the initial request invocations and dashed lines represent the response invocations.

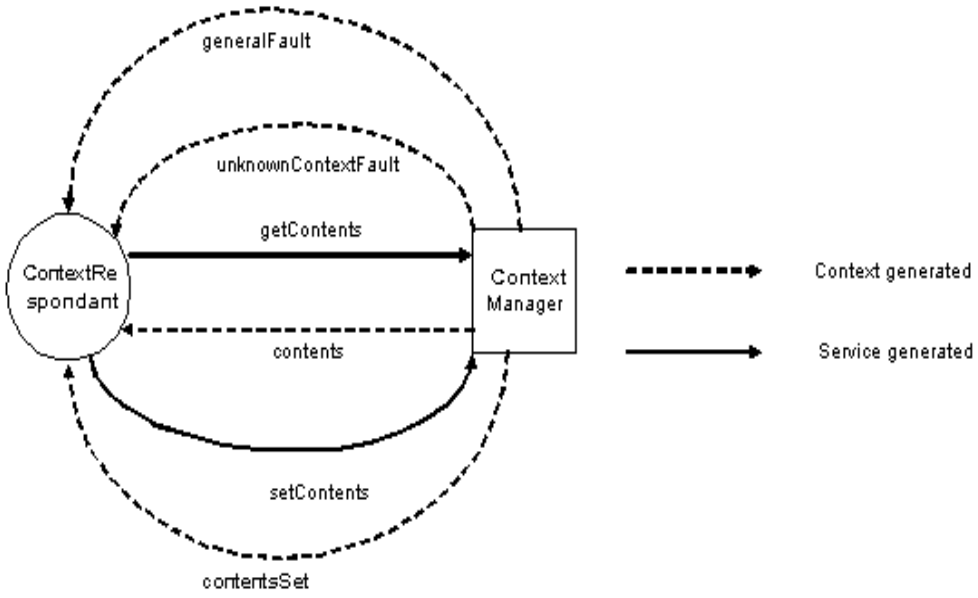

Figure 6, Context interactions.

The ContextManager has the following operations, all of which contain the call-back address for the ContextRespondant:

- *getContents*: this message is used to request the entire contents of a specific context.
- *setContents*: the contents of the context are replaced with the context information provided.

The ContextRespondant has the following operations:

- *contents*: this message returns the entire contents of a specific context.
- *contentsSet*: this message is sent to indicate that contents of the context have been updated.
- *unknownContextFault*: this message is sent to indicate that the specified context cannot be located.
- *generalFault*: this message is sent to indicate that some other error has occurred during the enlistment.

The WSDL interfaces that elucidate these roles are shown in Figure 7.

```
<wsdl:portType name="ContextManagerPortType">
  <wsdl:operation name="getContents">
    <wsdl:input message="tns:GetContentsMessage"/>
  </wsdl:operation>
  <wsdl:operation name="setContents">
    <wsdl:input message="tns:SetContentsMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="ContextRespondantPortType">
  <wsdl:operation name="contents">
    <wsdl:input message="tns:ContentsMessage"/>
```

```
  </wsdl:operation>
  <wsdl:operation name="contentsSet">
    <wsdl:input message="tns:ContentsSetMessage"/>
  </wsdl:operation>
  <wsdl:operation name="unknownContextFault">
    <wsdl:input message="tns:UnknownContextFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="generalFault">
    <wsdl:input message="tns:GeneralFaultMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

Figure 7, WSDL Interfaces for ContextManager and ContextRespondant Roles

Context information and SOAP

Where messages (either business-level application messages, or WS-CTX protocol messages themselves) require contextualization, the context is transported in a SOAP header block. While it is implicit that WS-CTX actors understand contexts that arrive in SOAP header blocks, the context propagated with application messages must also be understood by their recipients. Hence in this case each SOAP header block carrying an activity context has the “mustUnderstand” attribute set to “true” and the recipient must understand the header block encoding according to its identifying URI. This is shown in Figure 8.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2002/06/soap-envelope">
  <soap:Header
encodingStyle="http://www.webservicestransactions.org/schemas/wsctx/2003/03"
    mustUnderstand="true">
    <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>
</context>
  </soap:Header>
  <soap:Body>
    <!-- Application Payload -->
  </soap:Body>
</soap:Envelope>
```

Figure 8, CTX Context Transported in a SOAP Header Block

5. Context Service components

In this section we shall consider the individual components within the Context Service architecture. In order to support both synchronous request/response and message interactions, we shall describe the components in terms of their *behaviour and the interactions* that occur between them. All interactions are described in terms of correlated *messages*, which an implementation may abstract at a higher level into request/response pairs or RPCs, for example. As such, all communicated messages are required to contain response endpoint addresses solely for the purposes of each interaction, and a correlation identifier such that incoming and outgoing invocations can be associated.

Note that both addressing and message correlation are stop-gap measures which will be harmonized with the prevalent Web Services standards for addressing and reliable messaging once the community has reached consensus on appropriate standards.

One consequence of these interactions is that faults and errors which may occur when an service is invoked are communicated back to interested parties via messages which are themselves part of the standard protocol – and does not use the fault mechanisms of the underlying SOAP-based transport. 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 end-points are implemented and how they expose the various operations (e.g., via WSDL [1]) is not mandated by this specification.

5.1 Status

During the existence of the activity its status will either be running, completing, or completed. An activity should report its current status when asked; there is no notion of automatically informing services when a specific state is entered:

```
<xs:simpleType name="StatusType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="activity.status.ACTIVE"/>
    <xs:enumeration value="activity.status.COMPLETING"/>
    <xs:enumeration value="activity.status.COMPLETED"/>
    <xs:enumeration value="activity.status.NO_ACTIVITY"/>
    <xs:enumeration value="activity.status.UNKNOWN"/>
  </xs:restriction>
</xs:simpleType>
```

The meaning of each of the above values is given below:

- activity.status.ACTIVE: An Activity is associated with the target service and the Activity is in the active state. An implementation returns this status after an Activity has been started and prior to its beginning completion.
- activity.status.COMPLETING: An Activity is associated with the target service and it is in the process of completing. An implementation returns this status if it has started to complete, but has not yet finished the process. This value indicates that the activity may be performing activity specific work required to determine its final completion status, such as notifying participants of a failure. An activity must enter this state prior to completion.
- activity.status.COMPLETED: An Activity is associated with the target service and it has completed.
- activity.status.NO_ACTIVITY: No Activity is currently associated with the target service. This will occur after an Activity has completed, or before the first Activity is created.
- activity.status.UNKNOWN: An Activity is associated with the target service, but the Context Service cannot determine its current status. This is a transient condition, and a subsequent invocation should ultimately return a different status. An implementation may attempt to retry the appropriate invocation transparently if such a value is returned initially.

The diagram below indicates the transitions that an Activity can undergo.

Figure 9, Activity UML state diagram.

5.2 CompletionStatus

```
<xs:simpleType name="CompletionStatusType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="activity.complete.SUCCESS"/>
    <xs:enumeration value="activity.complete.FAIL"/>
    <xs:enumeration value="activity.complete.FAIL_ONLY"/>
    <xs:enumeration value="activity.complete.UNKNOWN"/>
  </xs:restriction>
</xs:simpleType>
```

When an Activity completes, it does so in one of two states, either *success* or *failure* (the meaning of which can only be defined by the application or services that use the activity). During its lifetime, the completion state of the Activity (i.e., the state it would have if it completed at that point) may change from success to failure, and back again many times. This is represented by the CompletionStatusType type, whose values are:

- activity.complete.SUCCESS: the Activity has successfully performed its work and can complete accordingly. When in this state, the Activity completion status can be changed.
- activity.complete.FAIL: some (application specific) error has occurred which has meant that the Activity has not performed all of its work, and should be driven during completion accordingly. When in this state, the Activity completion status can be changed.
- activity.complete.FAIL_ONLY: some (application specific) error has occurred which has meant that the Activity has not performed all of its work, and should be driven during completion accordingly. Once in this state, the completion status of the Activity cannot be changed, i.e., the only possible outcome for the Activity is for it to fail.
- activity.complete.UNKNOWN: The Context Service cannot determine its current completion status. This is a transient condition, and a subsequent invocation should ultimately return a different completion status. An implementation may attempt to retry the appropriate invocation transparently if such a value is returned initially.

5.3 Activity outcomes

When an Activity completes, an outcome may be returned to the initial Web service application in order for it to determine the final status of the Activity. Both success and failure states may be encoded within an outcome.

Services exchange messages based on the AssertionType defined in WS-CTX.

```
<xs:complexType name="AssertionType">
  <xs:sequence>
    <xs:element name="sender-address" type="tns:AddressType" minOccurs="0"/>
    <xs:element name="recipient-address" type="tns:AddressType" minOccurs="0"
      maxOccurs="0"/>
    <xs:element name="correlation-id" type="xs:string" minOccurs="0"/>
    <xs:any namespace="##other" processContents="lax" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

The type is designed to be extensible, returning a message containing arbitrary valid XML whose form and content is understood by the recipient.

5.4 ALS

The intention of the Context Service is to support the general notion of an activity and an associated context (including an activity identifier), with the capability for lifecycle services (ALS) to enroll with that Context Service and participate in the lifecycle of an activity, possibly enhancing the context if necessary. For example, as shown in Figure 10, a coordination service may be associated with the Context Service and customize the context to include information on the coordinator end-point(s) and participant(s). Note, the Context Service does not support coordination capabilities at all but simply maintains the execution environment-to-context repository.

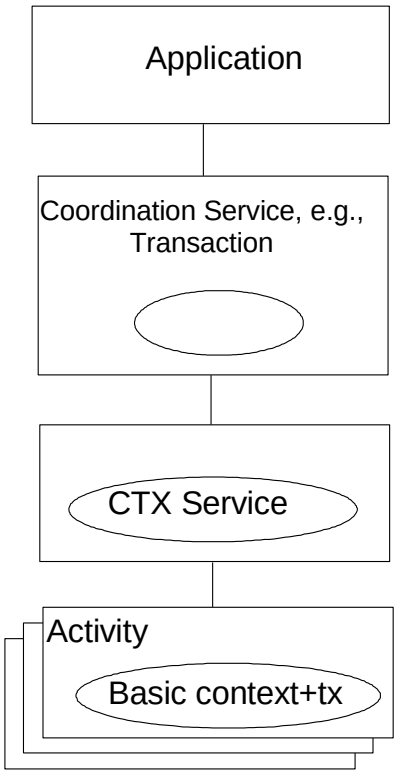

Figure 10, Context Service and coordination service relationship.

A specific combination of ALS-es maps to a specific type of context. For example, coordinator and transactions or just security. An Context Service may be responsible

for managing many different combinations of ALS-es and as such there is a requirement to be able to identify which ALS configuration is required. All ALS configurations are identified by an *ALS-configuration identifier*, which is a URI.

5.4.1 Contexts

All ALS contexts *must* be extensions of the basic Context Service context. Where messages require contextualization, the context is transported in a SOAP header block. There may be one context representing a number of different ALS-es or one context per ALS: this mapping is determined by the relationships between the ALS-es registered with a specific Context Service and is beyond the scope of this specification.

5.4.2 Enlisting and delisting an ALS

In order to enroll an ALS with the Context Service, the Context Service exposes the ALSRegistrar interface and the entity performing the enroll defines the ALSRegistrant interface to receive the call-back results of the enrolment attempt.

The ALSRegistrar has the following operation:

- *enlistALS*: the endpoint address for the ALS is specified in the message, along with the call-back address for the ALSRegistrant, which will be used by the ALSRegistrar to later send a messaging indicating success or failure. The ALS-configuration identifier is also present. Upon success, the ALS will be enrolled with all activities subsequently created by the Context Service instance that this ALSRegistrar represents.

The ALSRegistrant has the following operations, all of which contain the endpoint address for the ALS:

- *enlisted*: this message is sent by the ALSRegistrar to indicate that the ALS has been successfully enlisted.
- *invalidALSFault*: this message is sent to indicate that the specified ALS is invalid in the current Context Service.
- *generalFault*: this message is sent to indicate that some other error has occurred during the enlistment.

The message exchanges between the ALSRegistrar and the service that enlists the ALS is shown in Figure 11. As mentioned, this service receives protocol interactions via an ALSRegistrant service, but we have considered the two roles (enlister and receiver of enlistment messages from the ALSRegistrar) separately. This is purely for illustrative purposes and implementations are free to combine these roles if they see fit.

The service enrolling the ALS calls the ALSRegistrar *enlistALS* operation supplying the endpoint for the relevant ALS. The ALSRegistrar will eventually respond by calling either the *enlisted*, *invalidALSFault* or *generalFault* operations on the supplied ALSRegistrant service.

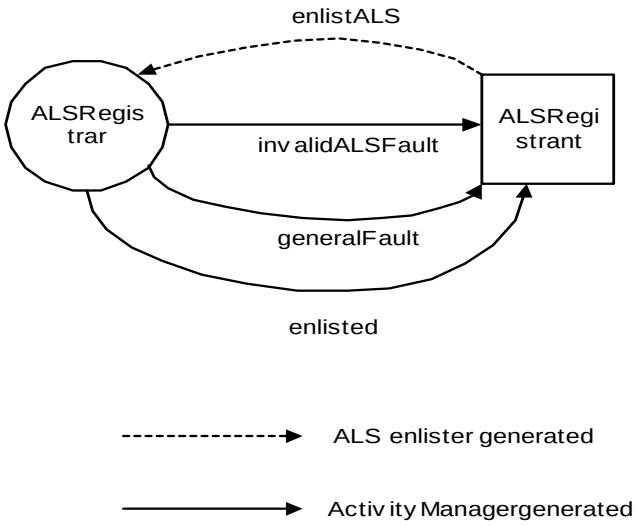

Figure 11, ALSRegistrar and ALSRegistrant interaction.

In order to de-list an ALS with the Context Service, the entity performing the de-list defines the ALSRegistrant interface to receive the results of the delistment attempt.

The ALSRegistrar has the following additional operation:

- *delistALS*: the endpoint address for the ALS is specified, along with the call-back address for the ALSRegistrant, which will be used by the ALSRegistrar to later call-back to indicate success or failure. The ALS-configuration identifier is also present. Upon success, the ALS will be un-enrolled from the Context Service instance that this ALSRegistrar represents.

The ALSRegistrant has the following operations, all of which contain the endpoint address for an ALS:

- *delisted*: this operation is called by the ALSRegistrar to indicate that the ALS has been successfully delisted.
- *invalidALSFault*: this operation is called to indicate that the specified ALS is invalid in the current Context Service – it was not enlisted.
- *generalFault*: this operation indicates that some other error has occurred during the enlistment.

The interactions between the ALSRegistrar and the service that delists the ALS is shown in Figure 12, highlighting the various message exchanges. As mentioned, this service receives protocol interactions via an ALSRegistrant service, but we have considered the two roles (delister and receiver of delistment messages from the ALSRegistrar) as separate. This is purely for illustrative purposes and implementations are free to combine these roles if they see fit.

The service delisting the ALS calls the ALSRegistrar *delistALS* operation supplying the endpoint for the relevant ALS.

The ALSRegistrar will eventually respond by calling either the *delisted*, *invalidALSFault* or *generalFault* operations on the supplied ALSRegistrant service.

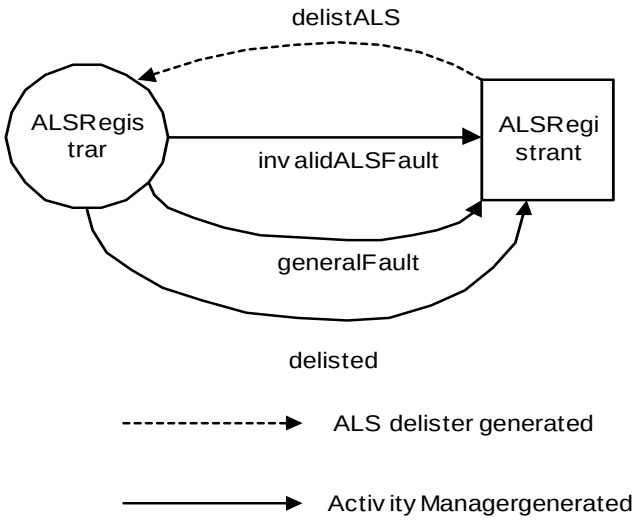

Figure 12, ALSRegistrar and ALSRegistrant interaction.

The WSDL describing the Registrar and Registrant roles is shown in Figure 13

```
<wsdl:portType name="ALSRegistrarPortType">
  <wsdl:operation name="enlistALS">
    <wsdl:input message="tns:EnlistALSMessage"/>
  </wsdl:operation>
  <wsdl:operation name="delistALS">
    <wsdl:input message="tns:DelistALSMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="ALSRegistrantPortType">
  <wsdl:operation name="enlistedALS">
    <wsdl:input message="tns:ALSEnlistedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="delistedALS">
    <wsdl:input message="tns:ALSDelistedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="invalidALSFault">
    <wsdl:input message="tns:InvalidALSFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="generalFault">
    <wsdl:input message="tns:GeneralFaultMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

Figure 13, ALSRegistrar and ALSRegistrant WSDL

5.4.3 ALS and Context Service interactions

As we discussed previously, any ALS that is registered with an Context Service is invoked during the lifecycle of each activity. The activity context is propagated to the ALS and the ALS may augment or manipulate the context as is appropriate for its

implementation. The ALS is also able to perform other implementation specific operations, e.g., coordination when the activity terminates.

In order to execute protocol interactions between Context Service and ALS, the following service interfaces are used:

- ALS: this represents the ALS endpoint in the protocol interaction.
- ALSRespondant: this represents the Context Service endpoint in the protocol interaction; this is where the ALS calls back with response messages.

The ALS has the following operations which the Context Service invokes. All of these operations are contextualised with the current context (if any) and the ALSRespondant endpoint is exchanged for call-backs:

- *begin*: This operation is invoked when a new activity has started.
- *completeWithStatus*: This operation is invoked when the activity is completing with the status provided. The service can perform whatever lifecycle management it requires (e.g., coordination) that is appropriate for the specified status.
- *complete*: The *complete* operation is called by the Context Service as part of the completion process of an activity to give the service the opportunity to perform any necessary clean-up work.
- *getIdentity*: this message is used to obtain the unique identification of a specific ALS.

The ALSRespondant has the following operations which the ALS invokes to indicate the result of a previous ALS invocation by the Context Service. All of these operations are contextualised with an augmented context (if any):

- *begun*: the ALS acknowledges that the activity has begun and may have performed implementation specific work.
- *completedWithStatus*: the ALS acknowledges that the activity is completing with the specified status. It may have performed implementation specific work.
- *completed*: the ALS acknowledges that the activity has completed. It may have performed implementation specific work.
- *identity*: this message is used to return the unique identification of a specific ALS.
- *invalidALSFault*: this operation is called to indicate that the specified ALS is invalid in the current Context Service.
- *validContextExpectedFault*: this operation is called to indicate that the ALS expected a valid context to be associated with the invoked operation but did not find one.
- *generalFault*: this operation indicates that some other error has occurred during the enlistment.

The interactions between the Context Service (through the ALSRespondant endpoint) and the ALS services is shown in Figure 14.

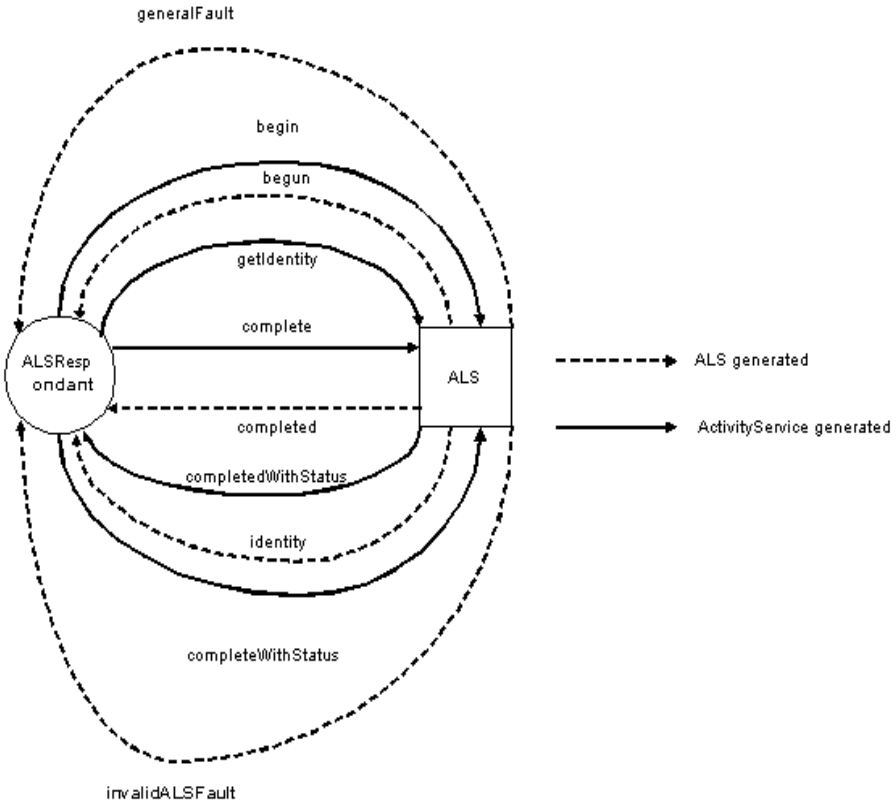

Figure 14, ALSRespondant and ALS interactions.

The Context Service invokes the ALS *begin* and the ALS will respond by calling back with either *begun*, *invalidALSFault* or *generalFault* on the ALSActivtyService.

If no faults are returned, the Context Service invokes the ALS *complete* or *completeWithStatus* and the ALS will eventually respond by calling back with either *completed*, *completedWithStatus*, *invalidALSFault* or *generalFault* on the ALSRespondant endpoint.

The Context Service invokes the ALS *complete* operation and the ALS will eventually respond by calling back with either *completed*, *invalidALSFault* or *generalFault* on the ALSRespondant endpoint.

```
<wsdl:portType name="ALSPortType">
  <wsdl:operation name="begin">
    <wsdl:input message="tns:ALSBeginMessage"/>
  </wsdl:operation>
  <wsdl:operation name="completeWithStatus">
    <wsdl:input message="tns:CompleteWithStatustMessage"/>
  </wsdl:operation>
  <wsdl:operation name="complete">
    <wsdl:input message="tns:CompleteMessage"/>
  </wsdl:operation>
  <wsdl:operation name="validContextExpectedFault">
    <wsdl:input message="tns:ValidContextExpectedFaultMessage"/>
```

```
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="ALSRespondantPortType">
  <wsdl:operation name="begun">
    <wsdl:input message="tns:BegunMessage"/>
  </wsdl:operation>
  <wsdl:operation name="completedWithStatus">
    <wsdl:input message="tns:CompletedWithStatusMessage"/>
  </wsdl:operation>
  <wsdl:operation name="completed">
    <wsdl:input message="tns:CompletedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="invalidALSFault">
    <wsdl:input message="tns:InvalidALSFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="validContextExpectedFault">
    <wsdl:input message="tns:ValidContextExpectedFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="generalFault">
    <wsdl:input message="tns:GeneralFaultMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

The Context Service WSDL

In order to be able to scope work within activities it is necessary for a component of the Context Service to provide an interface for activity demarcation. The implementation of this service may be collocated with users. Since the Context Service maintains information on multiple activities, an activity context may be present on all operation invocations to determine the appropriate activity on which to operate.

Interactions with the Context Service occur between users (services) and the Context Service via the UserCTXService and CTXService interfaces respectively. The WSDL for these services is shown below and we shall describe the interactions in the following section.

```
<wsdl:portType name="CTXServicePortType">
  <wsdl:operation name="begin">
    <wsdl:input message="tns:BeginMessage"/>
  </wsdl:operation>
  <wsdl:operation name="complete">
    <wsdl:input message="tns:CompleteMessage"/>
  </wsdl:operation>
  <wsdl:operation name="completeWithStatus">
    <wsdl:input message="tns:CompleteWithStatusMessage"/>
  </wsdl:operation>
  <wsdl:operation name="setCompletionStatus">
    <wsdl:input message="tns:SetCompletionStatusMessage"/>
  </wsdl:operation>
  <wsdl:operation name="getCompletionStatus">
```

```
    <wsdl:input message="tns:GetCompletionStatusMessage"/>
  </wsdl:operation>
  <wsdl:operation name="getStatus">
    <wsdl:input message="tns:GetStatusMessage"/>
  </wsdl:operation>
  <wsdl:operation name="getActivityName">
    <wsdl:input message="tns:GetActivityNameMessage"/>
  </wsdl:operation>
  <wsdl:operation name="getContext">
    <wsdl:input message="tns:GetContextMessage"/>
  </wsdl:operation>
  <wsdl:operation name="setTimeout">
    <wsdl:input message="tns:SetTimeoutMessage"/>
  </wsdl:operation>
  <wsdl:operation name="getTimeout">
    <wsdl:input message="tns:GetTimeoutMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="UserCTXServicePortType">
  <wsdl:operation name="begun">
    <wsdl:input message="tns:BegunMessage"/>
  </wsdl:operation>
  <wsdl:operation name="completed">
    <wsdl:input message="tns:CompletedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="completedWithStatus">
    <wsdl:input message="tns:CompletedWithStatusMessage"/>
  </wsdl:operation>
  <wsdl:operation name="completionStatusSet">
    <wsdl:input message="tns:CompletionStatusSetMessage"/>
  </wsdl:operation>
  <wsdl:operation name="completionStatus">
    <wsdl:input message="tns:CompletedWithStatusMessage"/>
  </wsdl:operation>
  <wsdl:operation name="completionStatus">
    <wsdl:input message="tns:CompletedWithStatusMessage"/>
  </wsdl:operation>
  <wsdl:operation name="status">
    <wsdl:input message="tns:StatusMessage"/>
  </wsdl:operation>
  <wsdl:operation name="activityName">
    <wsdl:input message="tns:ActivityNameMessage"/>
  </wsdl:operation>
  <wsdl:operation name="requestedContext">
    <wsdl:input message="tns:RequestedContextMessage"/>
  </wsdl:operation>
  <wsdl:operation name="timeoutSet">
    <wsdl:input message="tns:TimeoutSetMessage"/>
  </wsdl:operation>
  <wsdl:operation name="timeout">
    <wsdl:input message="tns:TimeoutMessage"/>
```

```
  </wsdl:operation>
  <wsdl:operation name="invalidStateFault">
    <wsdl:input message="tns:InvalidStateFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="invalidActivityFault">
    <wsdl:input message="tns:InvalidActivityFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="timeoutOutOfRange">
    <wsdl:input message="tns:TimeoutOutOfRangeFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="childActivityPendingFault">
    <wsdl:input message="tns:ChildActivityPendingFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="noActivityFault">
    <wsdl:input message="tns:NoActivityFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="noPermissionFault">
    <wsdl:input message="tns:NoPermissionFaultMessage"/>
  </wsdl:operation>
 </wsdl:portType>
```

In order to drive the Context Service, the following two roles (and associated services) are defined for the interactions:

- CTXService: this has operations begin, complete, completeWithStatus, setCompletionStatus, getCompletionStatus, getStatus, getActivityName, getContext, setTimeout and getTimeout;
- UserCTXService: this is the user/service callback endpoint address for the various CTXService operations. As such, it has operations begun, completed, completionStatusSet, completionStatus, status, activityName, context, timeoutSet, timeout, invalidStateFault, invalidActivityFault, timeoutOutOfRangeFault, childActivityPendingFault, noActivityFault, noPermissionFault, validContextExpectedFault.

The CTXService has the following operations, all of which are contextualised with the current context (if any) and contain the ALS-configuration identifier. The UserCTXService endpoint address is exchanged during each operation in order to allow the Context Service to return the result of the invocation.

begin

The *begin* operation creates a new activity and initializes the context. Any enrolled ALS Web services have their *begin* methods called in the scope of the newly initialized context and can augment it accordingly. A *begin* can be modelled as the first Web service in the Activity to register itself, in the case where there is no formal Context Service executing as a separate Web service; an unique activity identifier is created for the activity such that any context information that is subsequently obtained will reference this identifier. If an activity context is present on the *begin* request then the newly created Activity will be nested within it. Otherwise, the Activity exists at the top level. If the parent Activity has been marked as

activity.complete.FAIL_ONLY then the *invalidStateFault* operation will be invoked on the received UserCTXService endpoint. If the activity is completing, or has completed, the *invalidActivityFault* operation will be invoked on the received UserCTXService endpoint.

The timeout parameter is used to control the lifetime of the Activity. If the Activity has not completed by the time *timeout* seconds elapses then it is subject to being completed automatically by the Context Service with the activity.complete.FAIL status. The timeout can have the following possible values:

- *any positive value*: the Activity must complete within this number of seconds.
- -1: the Activity will never be completed automatically by the Context Service implementation, i.e., it will never be considered to have timed out.
- *0*: the last value specified using the set_timeout method is used. If no prior call to the *setTimeout* operation has occurred for this thread, or the value returned is 0, then it is implementation dependant as to the timeout value associated with this Activity.

Any other value results in the Context Service calling the *timeoutOutOfRangeFault* operation on the UserCTXService endpoint.

Upon success, the new activity will be placed in the activity.status.ACTIVE state and the Context Service will invoke the *begun* operation of the UserCTXService.

If an invalid context is propagated on the *begin* request then the *validContextExpectedFault* operation is invoked on the UserCTXService.

The *generalFault* operation is invoked on the UserCTXService if an unexpected error or fault occurs.

complete

A valid activity context is associated with this invocation. The *complete* operation causes the associated Activity to complete with its current CompletionStatus, or activity.complete.FAIL if none has been specified using *setCompletionStatus*. If there are any encompassed active Activities and the completion status is activity.complete.SUCCESS, then the *childActivityPendingFault* operation is invoked on the associated UserCTXService; the application must then either complete the outstanding nested contexts or force the Activity to end by setting the CompletionStatus to either activity.complete.FAIL or activity.complete.FAIL_ONLY and then call *complete* again. Prior to completing, any registered ALS is informed via the *completeWithStatus* method and is able to undertake its specific work. Subsequent to completion, their *complete* methods are invoked.

If the completion status is activity.complete.FAIL, or activity.complete.FAIL_ONLY, any encompassed active Activities will they have their completion status set to activity.complete.FAIL_ONLY. If there is no Activity associated with the sent context, the *noActivityFault* operation is invoked on the UserCTXService and no other action is taken. An Context Service implementation may impose restrictions on

which users can terminate an activity, and in which case the *noPermissionFault* operation may be invoked on the UserCTXService.

Once complete, the Context Service invokes the *completed* method on the UserCTXService, passing it the resultant AssertionType (which may be an empty message) and the state in which it completed; the AssertionType may be used to further interpret the final outcome of the Activity. If the Activity is not allowed to complete in the status required (e.g., it has been marked as activity.complete.FAIL_ONLY and is being asked to complete in activity.complete.SUCCESS state) then the Context Service will invoke the associated UserCTXServices *activityCompleted* operation. If the Activity has begun completion, or has completed, then the *invalidActivityFault* UserCTXService operation is called.

If an invalid context is propagated on the request then the *validContextExpectedFault* operation is invoked on the UserCTXService.

The *generalFault* operation is invoked on the UserCTXService if an unexpected error or fault occurs.

Additional fault or success messages are expected to be encoded within the resultant AssertionType.

completeWithStatus

A valid activity context should be associated with this invocation. The *completeWithStatus* operation causes the Activity associated with the context to complete and use the CompletionStatus provided if this does not conflict with any that has previously been set using the *setCompletionStatus* operation; this is logically equivalent to calling the *setCompletionStatus* operation followed by the complete operation. The UserCTXService *completed* operation is called upon success.

If an invalid context is propagated on the request then the *validContextExpectedFault* operation is invoked on the UserCTXService.

The *generalFault* operation is invoked on the UserCTXService if an unexpected error or fault occurs.

Additional fault or success messages are expected to be encoded within the resultant AssertionType.

setCompletionStatus

A valid activity context is associated with this invocation. This operation can be used to set the CompletionStatus that will be used when the Activity completes. This operation may be called many times during the lifetime of an Activity in order to reflect changes in its completion status as it executes. If this operation is not called during the Activity's lifetime, the default status is activity.complete.FAIL. When the Activity completes, the CompletionStatus is given to all enlisted ALS-es. It may also appear in the activity context. If the CompletionStatus is activity.complete.FAIL_ONLY and an attempt is made to change the status to anything other than activity.complete.FAIL_ONLY, the *invalidStateFault* operation

will be invoked on the UserCTXService. If the Activity has begun completion, or has completed, then the UserCTXService *invalidActivityFault* operation will be called. Otherwise, the *completionStatusSet* operation is invoked.

If an invalid context is propagated on the request then the *validContextExpectedFault* operation is invoked on the UserCTXService.

The *generalFault* operation is invoked on the UserCTXService if an unexpected error or fault occurs.

getCompletionStatus

This operation is used to obtain the current CompletionStatus associated with the activity referenced in the propagated context (if any). The Context Service will invoke the *completionStatus* on the UserCTXService associated with this invocation, passing the CompletionStatus currently associated with the Activity, or activity.complete.FAIL if *setCompletionStatus* has not previously been called. If there is no activity associated with the propagated context then the *noActivityFault* operation will be called.

If an invalid context is propagated on the request then the *validContextExpectedFault* operation is invoked on the UserCTXService.

The *generalFault* operation is invoked on the UserCTXService if an unexpected error or fault occurs.

getStatus

This operation is used to obtain the current Status of the activity referenced in the propagated context. The Context Service will invoke the *status* operation on the associated UserCTXService to return the current status of the Activity. If there is no Activity associated with the context, the *noActivityFault* operation is invoked on the UserCTXService.

If an invalid context is propagated on the request then the *validContextExpectedFault* operation is invoked on the UserCTXService.

The *generalFault* operation is invoked on the UserCTXService if an unexpected error or fault occurs.

getActivityName

This operation is used to retrieve the name of the activity referenced in the propagated context. The Context Service invokes the *activityName* operation on the UserCTXService passing back a printable string describing the activity, or an empty string if there is no activity associated with the invocation.

If an invalid context is propagated on the request then the *validContextExpectedFault* operation is invoked on the UserCTXService.

The *generalFault* operation is invoked on the UserCTXService if an unexpected error or fault occurs.

getContext

Given the unique activity identification, this operation causes the Context Service to invoke the *context* operation on the associated UserCTXService to returns the context URI for the Activity. If there is no activity associated with the identifier then the *noActivityFault* operation is used.

If an invalid context is propagated on the request then the *validContextExpectedFault* operation is invoked on the UserCTXService.

The *generalFault* operation is invoked on the UserCTXService if an unexpected error or fault occurs.

setTimeout

This operation modifies a state variable associated with the Context Service that affects the time-out period associated with the activities created by subsequent invocations of the *begin* operation. If the parameter has a non-zero value *n*, then activities created by subsequent invocations of *begin* will be subject to being completed if they do not complete before *n* seconds after their creation. The timeout can have the following possible values:

- *any positive value*: the Activity must complete within this number of seconds.
- -1: the Activity will never be completed automatically by the Context Service implementation, i.e., it will never be considered to have timed out.
- *0*: it is implementation dependant as to the meaning of passing 0 as the value.

A valid timeout value results in the Context Service calling the UserCTXService's *timeoutSet* operation. Any other value results in the *timeoutOutOfRangeFault* operation being invoked on the associated UserCTXService.

If an invalid context is propagated on the request then the *validContextExpectedFault* operation is invoked on the UserCTXService.

The *generalFault* operation is invoked on the UserCTXService if an unexpected error or fault occurs.

getTimeout

Upon successful execution, this operation causes the Context Service to invoke the timeout operation on the associated UserCTXService and return the value of the time-out period associated with activities created by calls to *begin*. This need not be the value associated with the current Activity, however.

If an invalid context is propagated on the request then the *validContextExpectedFault* operation is invoked on the UserCTXService.

The *generalFault* operation is invoked on the UserCTXService if an unexpected error or fault occurs.

Figure 15 shows the state transitions for an activity and how they relate to the various messages exchanges between the client/user of the Context Service and the Context

Service. As mentioned above, in order to participate in these message interactions the client/user supplies a UserCTXService endpoint.

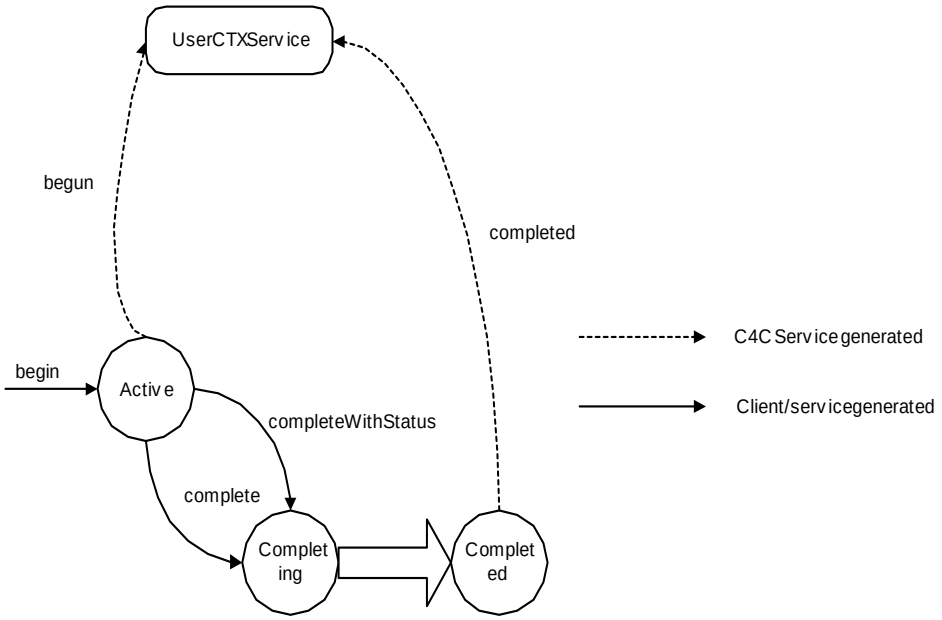

Figure 15, Activity state transitions and messages.

Context Service schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
targetNamespace="http://www.webservicestransactions.org/schemas/wsctx/2003/03"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://www.webservicestransactions.org/schemas/wsctx/2003/03"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <!-- Fundamental WSC4C types -->
  <xs:complexType name="ContextType">
    <xs:sequence>
      <xs:element name="context-identifier" type="xs:anyURI"/>
      <xs:element name="activity-service" type="xs:anyURI" minOccurs="0"/>
      <xs:element name="type" type="xs:anyURI" minOccurs="0"/>
      <xs:element name="activity-list" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="service" type="xs:anyURI" minOccurs="0"
maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute name="mustUnderstand" type="xs:boolean" use="optional"
default="false"/>
          <xs:attribute name="mustPropagate" type="xs:boolean" use="optional"
default="false"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="child-contexts" minOccurs="0">
```

```
        <xs:complexType>
          <xs:sequence>
            <xs:element name="child-context" type="tns:ContextType"
maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="timeout" type="xs:int" use="optional"/>
  </xs:complexType>
  <xs:element name="context" type="tns:ContextType"/>
  <xs:simpleType name="StatusType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="activity.status.ACTIVE"/>
      <xs:enumeration value="activity.status.COMPLETING"/>
      <xs:enumeration value="activity.status.COMPLETED"/>
      <xs:enumeration value="activity.status.NO_ACTIVITY"/>
      <xs:enumeration value="activity.status.UNKNOWN"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="status" type="tns:StatusType"/>
  <xs:simpleType name="CompletionStatusType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="activity.complete.SUCCESS"/>
      <xs:enumeration value="activity.complete.FAIL"/>
      <xs:enumeration value="activity.complete.FAIL_ONLY"/>
      <xs:enumeration value="activity.complete.UNKNOWN"/>
    </xs:restriction>
  </xs:simpleType>
  <!-- Message oriented elements -->
  <xs:complexType name="AddressType">
    <xs:sequence>
      <xs:element name="address" type="xs:anyURI"/>
      <!--      <xs:element name="portType" type="xs:QName"/> -->
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="AssertionType">
    <xs:sequence>
      <xs:element name="sender-address" type="tns:AddressType" minOccurs="0"/>
      <xs:element name="recipient-address" type="tns:AddressType" minOccurs="0"
maxOccurs="0"/>
      <xs:element name="correlation-id" type="xs:string" minOccurs="0"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="assertion" type="tns:AssertionType"/>
  <xs:complexType name="AssertionWithProtocolURIType">
    <xs:complexContent>
      <xs:extension base="tns:AssertionType">
```

```
        <xs:sequence>
          <xs:element name="protocol-uri" type="xs:anyURI"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="begin" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionWithProtocolURIType">
          <xs:sequence>
            <xs:element name="timeout" type="xs:int"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="begun" type="tns:AssertionType"
substitutionGroup="tns:assertion"/>
  <xs:element name="complete" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionWithProtocolURIType">
          <xs:sequence>
            <xs:element name="completion-status" type="tns:CompletionStatusType"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="completed" type="tns:AssertionType"
substitutionGroup="tns:assertion"/>
  <xs:element name="complete-with-status" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionWithProtocolURIType">
          <xs:sequence>
            <xs:element ref="tns:status"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="completed-with-status" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element name="completion-status" type="tns:CompletionStatusType"/>
          </xs:sequence>
        </xs:extension>
```

```
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="set-completion-status" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionWithProtocolURIType">
          <xs:sequence>
            <xs:element name="completion-status" type="tns:CompletionStatusType"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="get-completion-status" type="tns:AssertionWithProtocolURIType"
substitutionGroup="tns:assertion"/>
  <xs:element name="completion-status" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element name="completion-status" type="tns:CompletionStatusType"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="get-status" type="tns:AssertionWithProtocolURIType"
substitutionGroup="tns:assertion"/>
  <xs:element name="got-status" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element ref="tns:status"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="get-activity-name" type="tns:AssertionWithProtocolURIType"
substitutionGroup="tns:assertion"/>
  <xs:element name="activity-name" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element name="activity-name" type="xs:string"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
```

```
    </xs:complexType>
  </xs:element>
  <xs:element name="get-timeout" type="tns:AssertionWithProtocolURIType"
substitutionGroup="tns:assertion"/>
  <xs:element name="timeout" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element name="timeout" type="xs:int"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="set-timeout" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionWithProtocolURIType">
          <xs:sequence>
            <xs:element name="timeout" type="xs:int"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="timeout-set" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element name="timeout" type="xs:int"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="als-begin" type="tns:AssertionType"
substitutionGroup="tns:assertion"/>
  <xs:element name="get-context" type="tns:AssertionWithProtocolURIType"
substitutionGroup="tns:assertion"/>
  <xs:element name="requested-context">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element ref="tns:context"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
```

```
  </xs:element>
  <xs:element name="enlist-als" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionWithProtocolURIType">
          <xs:sequence>
            <xs:element name="als" type="xs:anyURI"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="als-enlisted" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element name="als" type="xs:anyURI"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="delist-als" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionWithProtocolURIType">
          <xs:sequence>
            <xs:element name="als" type="xs:anyURI"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="als-delisted" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element name="als" type="xs:anyURI"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="get-contents" type="tns:AssertionType"
substitutionGroup="tns:assertion"/>
  <xs:element name="contents" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
```

```
          <xs:sequence>
            <xs:element ref="tns:context"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="set-contents" substitutionGroup="tns:assertion">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element ref="tns:context"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="contents-set" type="tns:AssertionType"
substitutionGroup="tns:assertion"/>
  <!-- Faults -->
  <xs:complexType name="FaultType">
    <xs:complexContent>
      <xs:extension base="tns:AssertionType">
        <xs:sequence>
          <xs:element name="originator" type="xs:anyURI"/>
          <xs:element name="error-code" type="xs:anyURI"/>
          <xs:element name="description" type="xs:string" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="fault" type="tns:FaultType" abstract="true"/>
  <xs:element name="general-fault" type="tns:FaultType"
substitutionGroup="tns:fault"/>
  <xs:element name="invalid-als-fault">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:FaultType">
          <xs:sequence>
            <xs:element name="invalid-als-address" type="tns:AddressType"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="invalid-state-fault" type="tns:FaultType"
substitutionGroup="tns:fault"/>
  <xs:element name="invalid-activity-fault" type="tns:FaultType"
substitutionGroup="tns:fault"/>
  <xs:element name="timeout-out-of-range-fault" substitutionGroup="tns:fault">
```

```
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:FaultType">
          <xs:sequence>
            <xs:element name="specified-timeout" type="xs:int"/>
            <xs:element name="maximum-timeout" type="xs:int"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="child-activity-pending-fault" type="tns:FaultType"
substitutionGroup="tns:fault"/>
  <xs:element name="no-activity-fault" type="tns:FaultType"
substitutionGroup="tns:fault"/>
  <xs:element name="no-permission-fault" type="tns:FaultType"
substitutionGroup="tns:fault"/>
  <xs:element name="valid-context-expected-fault" type="tns:FaultType"
substitutionGroup="tns:fault"/>
  <xs:element name="unknown-context-fault" substitutionGroup="tns:fault">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:FaultType">
          <xs:sequence>
            <xs:element name="context-identifier" type="xs:anyURI"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Context Service WSDL Message Set

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://www.webservicestransactions.org/wsdl/wsctx/2003/03"
xmlns:tns="http://www.webservicestransactions.org/wsdl/wsctx/2003/03"
xmlns:wsc4c="http://www.webservicestransactions.org/schemas/wsctx/2003/03"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <wsdl:import location="http://www.webservicestransactions.org/schemas/wsctx/2003/03"
namespace="wsc4c.xsd"/>
  <wsdl:message name="ContextMessage">
    <wsdl:part name="content" element="wsc4c:context"/>
  </wsdl:message>
  <wsdl:message name="BeginMessage">
    <wsdl:part name="content" element="wsc4c:begin"/>
  </wsdl:message>
  <wsdl:message name="BegunMessage">
    <wsdl:part name="content" element="wsc4c:begun"/>
  </wsdl:message>
  <wsdl:message name="CompleteMessage">
    <wsdl:part name="content" element="wsc4c:complete"/>
  </wsdl:message>
  <wsdl:message name="CompletedMessage">
    <wsdl:part name="content" element="wsc4c:completed"/>
  </wsdl:message>
  <wsdl:message name="CompleteWithStatusMessage">
    <wsdl:part name="content" element="wsc4c:complete-with-status"/>
  </wsdl:message>
  <wsdl:message name="CompletedWithStatusMessage">
    <wsdl:part name="content" element="wsc4c:completed-with-status"/>
  </wsdl:message>
  <wsdl:message name="SetCompletionStatusMessage">
    <wsdl:part name="content" element="wsc4c:set-completion-status"/>
  </wsdl:message>
  <wsdl:message name="GetCompletionStatusMessage">
    <wsdl:part name="content" element="wsc4c:get-completion-status"/>
  </wsdl:message>
  <wsdl:message name="CompletionStatusMessage">
    <wsdl:part name="content" element="wsc4c:completion-status"/>
  </wsdl:message>
  <wsdl:message name="GetStatusMessage">
    <wsdl:part name="content" element="wsc4c:get-status"/>
  </wsdl:message>
  <wsdl:message name="StatusMessage">
    <wsdl:part name="content" element="wsc4c:got-status"/>
  </wsdl:message>
  <wsdl:message name="GetActivityNameMessage">
    <wsdl:part name="content" element="wsc4c:get-activity-name"/>
  </wsdl:message>
  <wsdl:message name="ActivityNameMessage">
```

```
    <wsdl:part name="content" element="wsc4c:activity-name"/>
  </wsdl:message>
  <wsdl:message name="GetTimeoutMessage">
    <wsdl:part name="content" element="wsc4c:get-timeout"/>
  </wsdl:message>
  <wsdl:message name="TimeoutMessage">
    <wsdl:part name="content" element="wsc4c:timeout"/>
  </wsdl:message>
  <wsdl:message name="SetTimeoutMessage">
    <wsdl:part name="content" element="wsc4c:set-timeout"/>
  </wsdl:message>
  <wsdl:message name="TimeoutSetMessage">
    <wsdl:part name="content" element="wsc4c:timeout"/>
  </wsdl:message>
  <wsdl:message name="ALSBeginMessage">
    <wsdl:part name="content" element="wsc4c:als-begin"/>
  </wsdl:message>
  <wsdl:message name="GetContextMessage">
    <wsdl:part name="content" element="wsc4c:get-context"/>
  </wsdl:message>
  <wsdl:message name="RequestedContextMessage">
    <wsdl:part name="content" element="wsc4c:requested-context"/>
  </wsdl:message>
  <wsdl:message name="EnlistALSMessage">
    <wsdl:part name="content" element="wsc4c:enlist-als"/>
  </wsdl:message>
  <wsdl:message name="ALSEnlistedMessage">
    <wsdl:part name="content" element="wsc4c:als-enlisted"/>
  </wsdl:message>
  <wsdl:message name="DelistALSMessage">
    <wsdl:part name="content" element="wsc4c:delist-als"/>
  </wsdl:message>
  <wsdl:message name="ALSDelistedMessage">
    <wsdl:part name="content" element="wsc4c:als-delisted"/>
  </wsdl:message>
  <wsdl:message name="GetContentsMessage">
    <wsdl:part name="content" element="wsc4c:get-contents"/>
  </wsdl:message>
  <wsdl:message name="ContentsMessage">
    <wsdl:part name="content" element="wsc4c:contents"/>
  </wsdl:message>
  <wsdl:message name="SetContentsMessage">
    <wsdl:part name="content" element="wsc4c:set-contents"/>
  </wsdl:message>
  <wsdl:message name="ContentsSetMessage">
    <wsdl:part name="content" element="wsc4c:contents-set"/>
  </wsdl:message>
  <wsdl:message name="GeneralFaultMessage">
    <wsdl:part name="content" element="wsc4c:general-fault"/>
  </wsdl:message>
  <wsdl:message name="InvalidALSFaultMessage">
```

```
    <wsdl:part name="content" element="wsc4c:invalid-als-fault"/>
  </wsdl:message>
  <wsdl:message name="InvalidStateFaultMessage">
    <wsdl:part name="content" element="wsc4c:invalid-state-fault"/>
  </wsdl:message>
  <wsdl:message name="InvalidActivityFaultMessage">
    <wsdl:part name="content" element="wsc4c:invalid-activity-fault"/>
  </wsdl:message>
  <wsdl:message name="TimoutOutOfRangeMessage">
    <wsdl:part name="content" element="wsc4c:timeout-out-of-range-fault"/>
  </wsdl:message>
  <wsdl:message name="ChildActivityPendingFaultMessage">
    <wsdl:part name="content" element="wsc4c:child-activity-pending-fault"/>
  </wsdl:message>
  <wsdl:message name="NoActivityFaultMessage">
    <wsdl:part name="content" element="wsc4c:no-activity-fault"/>
  </wsdl:message>
  <wsdl:message name="NoPermissionFaultMessage">
    <wsdl:part name="content" element="wsc4c:no-permission-fault"/>
  </wsdl:message>
  <wsdl:message name="ValidContextExpectedFaultMessage">
    <wsdl:part name="content" element="wsc4c:valid-context-expected-fault"/>
  </wsdl:message>
  <wsdl:message name="UnknownContextFaultMessage">
    <wsdl:part name="content" element="wsc4c:unknown-context-fault"/>
  </wsdl:message>
  <wsdl:portType name="ContextManagerPortType">
    <wsdl:operation name="getContents">
      <wsdl:input message="tns:GetContentsMessage"/>
    </wsdl:operation>
    <wsdl:operation name="setContents">
      <wsdl:input message="tns:SetContentsMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="ContextRespondantPortType">
    <wsdl:operation name="contents">
      <wsdl:input message="tns:ContentsMessage"/>
    </wsdl:operation>
    <wsdl:operation name="contentsSet">
      <wsdl:input message="tns:ContentsSetMessage"/>
    </wsdl:operation>
    <wsdl:operation name="unknownContextFault">
      <wsdl:input message="tns:UnknownContextFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="generalFault">
      <wsdl:input message="tns:GeneralFaultMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="CTXServicePortType">
    <wsdl:operation name="begin">
      <wsdl:input message="tns:BeginMessage"/>
```

```
    </wsdl:operation>
    <wsdl:operation name="complete">
      <wsdl:input message="tns:CompleteMessage"/>
    </wsdl:operation>
    <wsdl:operation name="completeWithStatus">
      <wsdl:input message="tns:CompleteWithStatusMessage"/>
    </wsdl:operation>
    <wsdl:operation name="setCompletionStatus">
      <wsdl:input message="tns:SetCompletionStatusMessage"/>
    </wsdl:operation>
    <wsdl:operation name="getCompletionStatus">
      <wsdl:input message="tns:GetCompletionStatusMessage"/>
    </wsdl:operation>
    <wsdl:operation name="getStatus">
      <wsdl:input message="tns:GetStatusMessage"/>
    </wsdl:operation>
    <wsdl:operation name="getActivityName">
      <wsdl:input message="tns:GetActivityNameMessage"/>
    </wsdl:operation>
    <wsdl:operation name="getContext">
      <wsdl:input message="tns:GetContextMessage"/>
    </wsdl:operation>
    <wsdl:operation name="setTimeout">
      <wsdl:input message="tns:SetTimeoutMessage"/>
    </wsdl:operation>
    <wsdl:operation name="getTimeout">
      <wsdl:input message="tns:GetTimeoutMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="UserCTXServicePortType">
    <wsdl:operation name="begun">
      <wsdl:input message="tns:BegunMessage"/>
    </wsdl:operation>
    <wsdl:operation name="completed">
      <wsdl:input message="tns:CompletedMessage"/>
    </wsdl:operation>
    <wsdl:operation name="completedWithStatus">
      <wsdl:input message="tns:CompletedWithStatusMessage"/>
    </wsdl:operation>
    <wsdl:operation name="completionStatusSet">
      <wsdl:input message="tns:CompletionStatusSetMessage"/>
    </wsdl:operation>
    <wsdl:operation name="completionStatus">
      <wsdl:input message="tns:CompletedWithStatusMessage"/>
    </wsdl:operation>
    <wsdl:operation name="completionStatus">
      <wsdl:input message="tns:CompletedWithStatusMessage"/>
    </wsdl:operation>
    <wsdl:operation name="status">
      <wsdl:input message="tns:StatusMessage"/>
    </wsdl:operation>
```

```
    <wsdl:operation name="activityName">
      <wsdl:input message="tns:ActivityNameMessage"/>
    </wsdl:operation>
    <wsdl:operation name="requestedContext">
      <wsdl:input message="tns:RequestedContextMessage"/>
    </wsdl:operation>
    <wsdl:operation name="timeoutSet">
      <wsdl:input message="tns:TimeoutSetMessage"/>
    </wsdl:operation>
    <wsdl:operation name="timeout">
      <wsdl:input message="tns:TimeoutMessage"/>
    </wsdl:operation>
    <wsdl:operation name="invalidStateFault">
      <wsdl:input message="tns:InvalidStateFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="invalidActivityFault">
      <wsdl:input message="tns:InvalidActivityFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="timeoutOutOfRange">
      <wsdl:input message="tns:TimeoutOutOfRangeFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="childActivityPendingFault">
      <wsdl:input message="tns:ChildActivityPendingFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="noActivityFault">
      <wsdl:input message="tns:NoActivityFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="noPermissionFault">
      <wsdl:input message="tns:NoPermissionFaultMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="ALSPortType">
    <wsdl:operation name="begin">
      <wsdl:input message="tns:ALSBeginMessage"/>
    </wsdl:operation>
    <wsdl:operation name="completeWithStatus">
      <wsdl:input message="tns:CompleteWithStatustMessage"/>
    </wsdl:operation>
    <wsdl:operation name="complete">
      <wsdl:input message="tns:CompleteMessage"/>
    </wsdl:operation>
    <wsdl:operation name="validContextExpectedFault">
      <wsdl:input message="tns:ValidContextExpectedFaultMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="ALSRespondantPortType">
    <wsdl:operation name="begun">
      <wsdl:input message="tns:BegunMessage"/>
    </wsdl:operation>
    <wsdl:operation name="completedWithStatus">
      <wsdl:input message="tns:CompletedWithStatusMessage"/>
```

```
    </wsdl:operation>
    <wsdl:operation name="completed">
      <wsdl:input message="tns:CompletedMessage"/>
    </wsdl:operation>
    <wsdl:operation name="invalidALSFault">
      <wsdl:input message="tns:InvalidALSFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="validContextExpectedFault">
      <wsdl:input message="tns:ValidContextExpectedFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="generalFault">
      <wsdl:input message="tns:GeneralFaultMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="ALSRegistrarPortType">
    <wsdl:operation name="enlistALS">
      <wsdl:input message="tns:EnlistALSMessage"/>
    </wsdl:operation>
    <wsdl:operation name="delistALS">
      <wsdl:input message="tns:DelistALSMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="ALSRegistrantPortType">
    <wsdl:operation name="enlistedALS">
      <wsdl:input message="tns:ALSEnlistedMessage"/>
    </wsdl:operation>
    <wsdl:operation name="delistedALS">
      <wsdl:input message="tns:ALSDelistedMessage"/>
    </wsdl:operation>
    <wsdl:operation name="invalidALSFault">
      <wsdl:input message="tns:InvalidALSFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="generalFault">
      <wsdl:input message="tns:GeneralFaultMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <!-- SOAP 1.1 over HTTP bindings -->
  <wsdl:binding name="ContextManagerPortTypeSOAPBinding"
type="tns:ContextManagerPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="getContents">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/getContents"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="setContents">
```

```
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/setContents"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="ContextRespondandtPortTypeSOAPBinding"
type="tns:ContextRespondandtPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="contents">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/contents"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="contentsSet">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/contentsSet"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="unknownContextFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/unknownContextFa
ult" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="unknownContextFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/unknownContextFa
ult" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="CTXServicePortTypeSOAPBinding" type="tns:CTXServicePortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="begin">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/begin"
style="document"/>
      <wsdl:input>
```

```
        <soap:body use="literal"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="complete">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/complete"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="completeWithStatus">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/completeWithStat
us" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="setCompletionStatus">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/setCompletionSta
tus" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="getCompletionStatus">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/getCompletionSta
tus" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="getStatus">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/getStatus"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="getActivityName">
```

```
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/getActivityName"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="getContext">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/getContext"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="setTimeout">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/setTimeout"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="getTimeout">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/getTimeout"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="begun">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/begun"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="completedWithStatus">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/completedWithSta
tus" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
```

```
    </wsdl:operation>
    <wsdl:operation name="completed">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/completed"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="generalFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/generalFault"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="invalidALSFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/invalidALSFault"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="UserCTXServicePortTypeSOAPBinding"
type="tns:UserCTXServicePortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="begun">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/begun"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="completed">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/completed"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="completedWithStatus">
```

```
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/completedWithSta
tus" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="completionStatusSet">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/completedWithSta
tus" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="completionStatus">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/completionStatus
" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="completionStatus">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/completionStatus
" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="status">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/status"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="activityName">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/activityName"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
```

```
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="requestedContext">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/requestedContext
" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="timeoutSet">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/timeoutSet"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="timeout">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/timeout"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="invalidStateFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/invalidStateFaul
t" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="invalidActivityFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/invalidActivityF
ault" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="timeoutOutOfRange">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/timeoutOutOfRang
e" style="document"/>
```

```
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="childActivityPendingFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/childActivityPen
dingFault" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="noActivityFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/noActivityFault"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="noPermissionFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/noPermissionFaul
t" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="ALSRegistrarPortTypeSOAPBinding"
type="tns:ALSRegistrarPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="enlistALS">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/enlistALS"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="delistALS">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/delistALS"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
```

```
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="ALSRegistrantPortTypeSOAPBinding"
type="tns:ALSRegistrantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="enlistedALS">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/enlistedALS"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="delistedALS">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/delistedALS"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="invalidALSFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/invalidALSFault"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="generalFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/generalFault"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="ALSPortTypeSOAPBinding" type="tns:ALSPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="begin">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/begin"
style="document"/>
      <wsdl:input>
```

```
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="completeWithStatus">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/completeWithStat
us" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="validContextExpectedFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/validContextExpe
ctedFault" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="complete">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/complete"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="ALSRespondantPortTypeSOAPBinding"
type="tns:ALSRespondantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="begun">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/begun"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="completedWithStatus">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/completedWithSta
tus" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
```

```
    </wsdl:operation>
    <wsdl:operation name="completed">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/completed"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="invalidALSFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/invalidALSFault"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="validContextExpectedFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/validContextExpe
ctedFault" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="generalFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wsctx/2003/03/generalFault"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

6. References

[1] WSDL 1.1 Specification, see http://www.w3.org/TR/wsdl

7. Acknowledgements

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

Dave Ingham, Arjuna Technologies Ltd.

Barry Hodgson, Arjuna Technologies Ltd.

Goran Olsson, Oracle Corporation.

Nickolas Kavantzas, Oracle Corporation.

Aniruddha Thankur, Oracle Corporation.