Web Services Transaction Management (WS-TXM) 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
An increasing number of applications are being constructed by combining or coordinating the execution of multiple Web services, each of which may represent an interface to a different underlying technology. The resulting applications can be very complex in structure, with complex relationships between their constituent services. 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 services requiring user interactions. In the loosely coupled environment represented by Web services, long running applications will require support for recovery and compensation, because machines may fail, processes may be cancelled, or services may be moved or withdrawn. Web services transactions also must span multiple transaction models and protocols native to the underlying technologies onto which the Web services are mapped.
A common technique for fault-tolerance is through the use of atomic transactions, which have the well know ACID properties, operating on persistent (long-lived) objects. Transactions ensure that only consistent state changes take place despite concurrent access and failures. However, traditional transactions depend upon tightly coupled protocols, and thus are often not well suited to more loosely-coupled Web services based applications, although they are likely to be used in some of the constituent technologies. It is more likely that traditional transactions are used in the minority of cases in which the cooperating Web services can take advantage of them, while new mechanisms, such as compensation, replay, and persisting business process state, more suited to Web services are developed and used for the more typical case.
WS-TXN provides a suite of transaction models, each suited to solving a different problem domain. However, because WS-TXN leverages WS-CF, it is intended to allow flexibility in the types of models supported. Therefore, if new models are required for other problem areas, they can be incorporated within this specification.

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

Atomic transactions are a well-known technique for guaranteeing consistency in the presence of failures. The ACID properties of atomic transactions (Atomicity, Consistency, Isolation, and Durability) ensure that even in complex business applications consistency of state is preserved, despite concurrent accesses and failures. This is an extremely useful fault-tolerance technique, especially when multiple, possibly remote, resources are involved.

The concepts of atomic transactions have played a cornerstone role in creating today's enterprise application environments by providing guaranteed consistent outcome in complex multiparty business operations and a useful separation of concerns in applications. While numerous multiparty business applications involve various patterns based on atomic transactions in order to solve non-trivial business problems, it was not until recently the word "business transactions" accumulated any concrete meaning. Rapid developments in Internet infrastructure and protocols have yielded a new type of application interoperation concept that makes concepts which could only previously be considered in an abstract form an implementation reality. The effects of such changes have been felt most strongly in business environments, fuelling the mindset for a transition from traditional atomic transactions to extended transaction models better suited for Internet interoperation.

Most business-to-business applications require transactional support in order to guarantee consistent outcome and correct execution. These applications often involve long running computations, loosely coupled systems and components that do not share data, location, or administration and it is thus difficult to incorporate traditional ACID transactions within such architectures. For example, an airline reservation system may reserve a seat on a flight for an individual for a specific period of time, but if the individual does not confirm the seat within that period it will be unreserved.

The structuring mechanisms available within traditional transaction systems are sequential and concurrent composition of transactions. These mechanisms are sufficient if an application function can be represented as a single top-level transaction. Frequently with Web services this is not the case. Top-level transactions are most suitably viewed as "short-lived" entities, performing stable state changes to the system; they are less well suited for structuring "long-lived" application functions (e.g., running for minutes**,** hours, days, …). Long-lived top-level transactions implemented using traditional systems may reduce the concurrency in the system to an unacceptable level by holding on to locks for a long time; further, if such a transaction rolls back, much valuable work already performed could be undone. Web services, because of their inherently unpredictable invocation patterns do not fit well with traditional ACID systems.

1.1 Problem statement

As Web Services have evolved as a means to integrate processes and applications at an both inside and outside the firewall, and as Web technologies have become firmly established and widely adopted, traditional transaction semantics and protocols have proven to be inappropriate for some Web services-based applications and services.

These particular Web services-based transactions differ from traditional transactions in that they execute over long periods, they require commitments to the transaction to be “negotiated” at runtime, and isolation levels have to be relaxed.

Structuring certain activities from long-running transactions can reduce the amount of concurrency within an application or (in the event of failures) require work to be performed again. For example, there are certain classes of application where it is known that resources acquired within a transaction can be released “early”, rather than having to wait until the transaction terminates; in the event of the transaction rolling back, however, certain compensation activities may be necessary to restore the system to a consistent state. Such compensation activities (which may perform forward or backward recovery) will typically be application specific, may not be necessary at all, or may be more efficiently dealt with by the application.

The goals of the specification are to:

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

2. Architecture

WS-TXM leverages the WS-CF and WS-CTX specifications. Figure 4 illustrates the layering of WS-TXM onto WS-CF. WS-TXM defines a pluggable transaction protocol that can be used with the coordinator to negotiate a set of actions for all participants to execute based on the outcome of a series of related Web services executions. The executions are related through the use of shared context. Examples of coordinated outcomes include the classic two-phase commit protocol, a three phase commit protocol, open nested transaction protocol, asynchronous messaging protocol, or business process automation protocol.

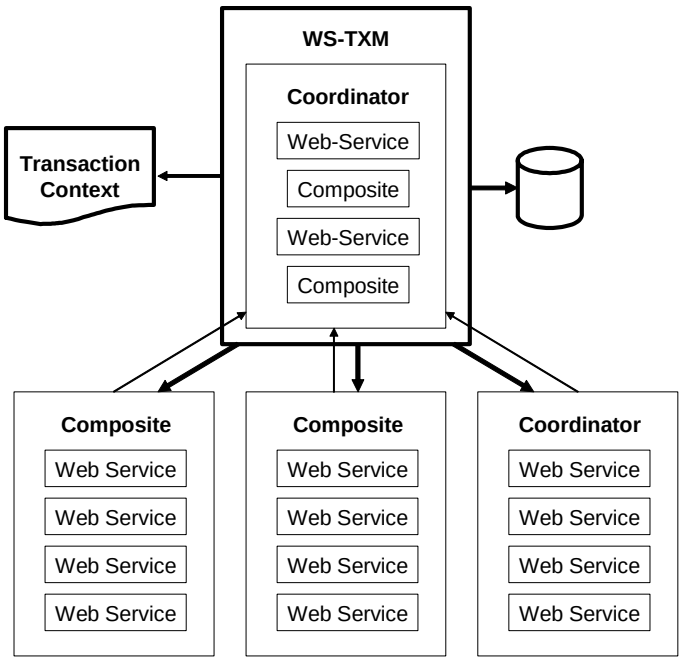

Figure 1, Relationship of transactions to coordination framework.

Coordinators can be participants of other coordinators, as shown above. When a coordinator registers itself with another coordinator, it can represent a series of local activities and map a neutral transaction protocol onto a platform-specific transaction 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.

3. Use case scenarios

In this section we shall briefly describe some of the use cases we believe that WS-TXM can address.

3.1 Web services coordination

Web services will typically not open up two-phase commit protocols to be driven by external coordinators, and may often combine disparate underlying technologies into a larger unit of work. As a result, coordinating multiple Web services within a single transaction can *never* give the same ACID guarantees as multiple two-phase commit resources: in a single-phase model if a failure occurs after having committed some

resources it is not possible to undo those that have committed. There are two solutions to this:

1) Wrap these one-phase objects in a two-phase wrapper which ignores the "prepare" phase and register them with a traditional transaction manager.
2) Treat this as an extended transaction model that requires specific coordination and error treatment.

In terms of implementation, there is no difference between 1 and 2: the resources are still one phase and may fail in exactly the same manner. However, on a conceptual level there is a significant difference: when using a transaction manager, programmers expect an all-or-nothing effect, especially since applications typically do not know about one-phase/two-phase restrictions of resources and may mix them in the same transaction; raising heuristic exceptions is the only possible solution for the transaction manager that finds it cannot undo committed operations, and then the application (or typically the administrator) has to deal with the outcome.

From the point of view of the user, the essential information is the state of the system after recovery from failure. The traditional transaction model allowed the users to easily understand the state of the data management resources as of the last successful transaction. In a long running, mixed model world it's equally important for the users to discover how far the transaction progressed, but the mechanisms and level of human interaction required to recover the situation may be very different. In the world of Web services transactions it's less likely that automatic recovery will be possible in as many cases as in a more tightly controlled environment, and more likely that human intervention will need to play a bigger part in restoring the system to normalcy.

Giving applications a specific extended transaction model that clearly defines how resources behave *and* does not allow one-phase and two-phase resources to be registered in the same transaction, gives a better understanding to users: the programmer must make a conscious choice as to the model that is being used, and the entire application is structured accordingly. In the end these types of applications are not transactional, and a traditional two-phase aware transaction system is inappropriate for them.

This notion of having a different extended transaction model for each use-case makes applications aware of the issues involved and helps to categorize objects and activities into which type of model they support. So, for example, one object may be used successfully within a two-phase and one-phase model, whereas another may only be used within two-phase. It is important to reduce complexity in developing "transactional" internet application by not over overloading a given model (e.g., ACID transactions).

It is important for users to be able to discover at any point during the execution of a business process exactly how far the transactional aspect of the process has progressed, and to be given options for branching or redirecting the outcome in the face of potentially recoverable errors.

3.2 Timed transactions

Many business transactions have specific "real-time" deadlines within which they must operate (e.g., purchasing of shares). After the deadline has elapsed, if the business transaction has not completed in a normal manner, there will typically be application specific ways in which it must terminate (e.g., purchase the shares at the current price if it is less than a certain value).

3.3 Arranging a night out

Consider the following long running business transaction, illustrated by Figure 2. The application activity is concerned with booking a taxi (*t1*), reserving a table at a restaurant (*t2*), reserving a seat at the theatre (*t3*), and then booking a room at a hotel (*t4*). If all of these operations were performed as a single transaction (shown by the dotted ellipse), then resources acquired during *t1* would not be released until the top-level transaction has terminated. If subsequent activities *t2*, *t3* etc. do not require those resources, then they will be needlessly unavailable to other clients.

Long-running applications and activities can be structured as many independent, short-duration top-level transactions, to form a long-running business transaction. This structuring allows an activity to acquire and use resources for only the required duration of this long-running transactional activity. Therefore, as shown the business transaction may be structured as many different, coordinated, short-duration top-level transactions.

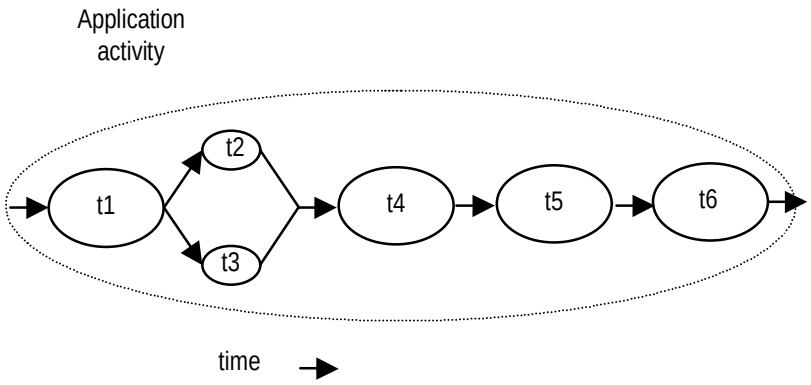

Figure 2, An example of a logical long-running "transaction", without failure.

However, if failures and concurrent access occur during the lifetime of these individual transactional activities then the behaviour of the entire long-running transaction may not possess ACID properties. Therefore, some form of (application specific) compensation may be required to attempt to return the state of the system to (application specific) consistency. Just as the application programmer has to implement the transactional work in the non-failure case, so too will programmers typically have to implement compensation transactions, since only they have the necessary application specific knowledge. Note, for simple or well-ordered work it is possible to provide automatic compensations.

For example, let us assume that *t4* has failed (rolls back). Further assume that the application can continue to make forward progress, but in order to do so must now

undo some state changes made prior to the start of *t4* (by *t1, t2* or *t3*). Therefore, new activities are started; *tc1* which is a compensation activity that will attempt to undo state changes performed, by say *t2*, and *t3* which will continue the application once *tc1* has completed. *tc5'* and *tc6'* are new activities that continue after compensation, e.g., since it was not possible to reserve the theatre, restaurant and hotel, it is decided to book tickets at the cinema. Obviously other forms of transaction composition are possible.

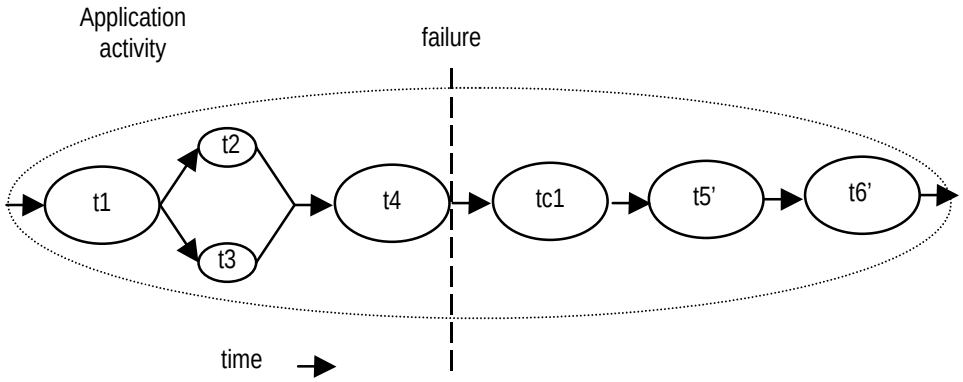

Figure 3, An example of a logical long-running "transaction", with failure.

It should be noted that even with suitable compensations, it can never be guaranteed to make the entire activity transactional: in the time between the original transaction completing and its compensation running, other activities may have performed work based upon the results of the yet to be compensated transaction. Attempting to undo these additional transactions (if this is possible) can result in an avalanche of compensations that may still not be able to return the system to the state it had prior to the execution of the first transaction. In addition, compensations may (continually) fail and it will then be extremely important to inform users (or system administrators). In the world of Web services transactions, automatic recovery is by definition less possible than previously.

Note, it will be application specific as to whether or not compensation should be tried again if it does fail. For example, consider the situation where a transaction sells shares and the compensation is to buy them back; if the compensation fails it may be inappropriate (and expensive) to try it again until it does eventually succeed if the share price is going up rapidly.

3.4 Home entertainment system

Let us assume that we are interested in building our own customized home entertainment system consisting of TV, DVD player, hi-fi and video recorder. Furthermore, rather than purchase each of these from the same manufacturer we want to shop around and get the best of each from possibly different sources.

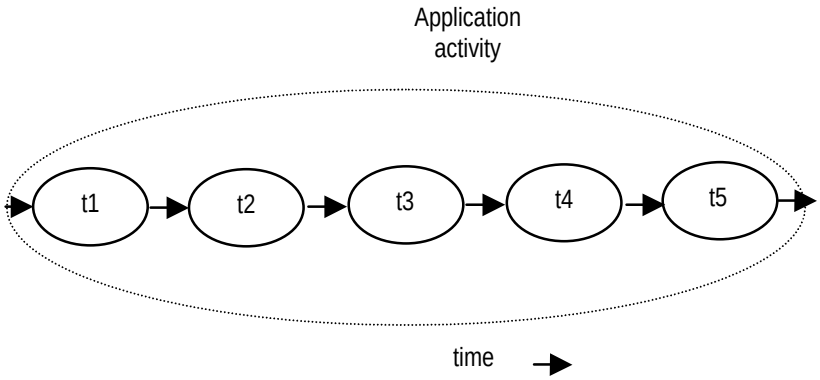

Figure 4, Building a home entertainment system via the Web.

When we visit the TV site we wish to start a transactional activity, t1, that will allow us to search and provisionally reserve (obtain transactional locks on) a number of different televisions (set A) that match our requirements. Before t1 terminates, we select a subset of the televisions, B, we are interested in, and provide a reference to our online bank account which the television site may contact to check that we have sufficient funds. t1 then ends, any locks obtained that are not members of B are released as normally for a transaction, (allowing other users to acquire them immediately if necessary), and all other locks are obtained and passed to t2.

The sequence of operations for t2, t3, and t4 are identical, where only subsets of items (DVD player, hi-fi and video) we have transactionally locked is released when each activity terminates. By the time t5 is executed there is a list of items that are locked and under its control, possibly also including the on-line bank account. Therefore, t5 is responsible for committing or rolling back the final purchase order. All locked resources are atomically handed off to the next atomic action, and failures do not require compensation: the atomicity property of t1, t2, t3, t4 and t5 ensures that either the purchase happens, or it does not (and all resources will be released).

4. Web Services transaction management

The WS-TXM specification defines three transaction models that address different use cases in current business-to-business interactions. However, it is likely that other transaction models will be required for different problem domains and therefore this specification is intended to incorporate these other models when required. There is a variety of models for Web services transactions depending on their use – 1PC, 2PC, distributed 2PC with interoperability, business process coordination. As such, it is the intention of the specification authors and supporters that other models should be added as and when they become available.

- *ACID transaction*: a traditional ACID transaction (AT) designed for interoperability across existing transaction infrastructures.

- *Long running action*: an activity, or group of activities, which does not necessarily possess the guaranteed ACID properties. A long running action (LRA) still has the "all or nothing" atomic effect, i.e., failure should not result in partial work. Participants within an LRA may use forward (compensation) or backward error recovery to ensure atomicity. Isolation is also considered a back-end implementation responsibility.
- *Business process transaction*: an activity, or group of activities, that is responsible for performing some application specific work. A business process (BP) may be structured as a collection of atomic transactions or long running actions depending upon the application requirements.

Because WS-TXM uses WS-CF, it builds upon the context information defined by that specification. Each transaction protocol specified within WS-TXM has its own context which will be described in the relevant sections.

4.1 Relationship to WS-CTX and WS-CF

WS-TXM builds on the Web Services Coordination Framework (WS-CF) and Web Service CTX Service (WS-CTX) specifications. It does this by defining specific coordinator and participant services and augmenting the distribution context.

In order to support the three transaction models, WS-TXM imposes the following restrictions on its interactions with WS-CF:

- The ACID transaction, long running action and business process models bind the scope of an activity to the scope of a "transaction". This is similar to the way that WS-CF binds the scope of an activity to the lifetime of a coordinator.
- In the rest of this text we shall assume that participants for the various transaction models are instance of the WS-CF Participant service. However, because that aspect of the WS-CF specification is optional, the Participant services may be defined elsewhere. Only the message exchanges are mandated for interoperability.

4.2 ACID transactions

The *ACID transaction* model recognizes that Web Services are for interoperability as much as for the Internet. As such, interoperability of existing transaction processing systems will be an important part of Web Services Transaction Management: such systems already form the backbone of enterprise level applications and will continue to do so for the Web Services equivalent. Business-to-business activities will typically involve back-end transaction processing systems either directly or indirectly and being able to tie together these environments will be the key to the successful take-up of Web Services transactions.

Although ACID transactions may not be suitable for all Web Services, they are most definitely suitable for some, and particularly high-value interactions such as those involved in finance. As a result, the ACID transaction model defined in WS-TXM has been designed with interoperability in mind.

In the ACID model, each activity is bound to the scope of a transaction, such that the end of an activity automatically triggers the termination (commit or rollback) of the associated transaction. The coordinator-type URI for the ACID transaction model is http://www.webservicestransactions.org/wsdl/wstxm/tx-acid/2003/03

4.2.1 Restrictions imposed on using WS-CF

As well as the restrictions outlined previously for general WS-TXM protocols, the ACID transaction model imposes the following additional restrictions:

- It is illegal to attempt to remove a participant from a transaction at any time. When the transaction terminates, participants are implicitly removed. As such, any attempt to call *removeParticipant* will result in the *wrongState* message being sent by the coordinator.
- It is illegal to call the WS-CF *coordinate* operation on the coordinator. All WS-TXM coordinator implementations will return the *notCoordinated* message if the coordinator receives a *coordinate* request.

4.2.2 Two-phase commit

The ACID transaction model uses a traditional two-phase commit protocol [2] with the following optimizations:

- *Presumed rollback*: the transaction coordinator need not record information about the participants in stable storage until it decides to commit, i.e., until after the prepare phase has completed successfully.
- *One-phase*: if there is only a single participant involved in the transaction then there is no need for a prepare phase since consensus is implicit.
- *Read-only:* a participant that is responsible for a service that did not modify any transactional data during the course of the transaction can indicate to the coordinator during prepare that it is a *read-only participant* and it will be omitted from the second phase of the commit protocol.

Participants that have successfully passed the *prepare* phase are allowed to make autonomous decisions as to whether they commit or rollback. A participant that makes such an autonomous choice *must* record its decision in case it is eventually contacted to complete the original transaction. If the coordinator eventually informs the participant of the fate of the transaction and it is the same as the autonomous choice the participant made, then there is obviously no problem: the participant simply got there before the coordinator did. However, if the decision is contrary, then a non-atomic outcome has happened: a *heuristic outcome*, with a corresponding *heuristic decision*.

The possible heuristic outcomes are:

- *Heuristic rollback*: the commit operation failed because some or all of the participants unilaterally rolled back the transaction.

- *Heuristic commit*: an attempted rollback operation failed because all of the participants unilaterally committed. This may happen if, for example, the coordinator was able to successfully prepare the transaction but then decided to roll it back (e.g., it could not update its log) but in the meanwhile the participants decided to commit.
- *Heuristic mixed*: some updates were committed while others were rolled back.
- *Heuristic hazard*: the disposition of some of the updates is unknown. For those which are known, they have either all been committed or all rolled back.

4.2.3 Coordinator state transitions for two-phase commit protocol

As shown in Figure 5, and in line with the basic WS-CTX, when the activity begins a *begin* message is sent by the activity service to the ACID transaction protocol ALS (Tx ALS); this then creates a corresponding coordinator that is associated with the activity through the context. The coordinator begins in the *Active* state and has the lifetime period associated with the activity. What this means is that if the activity timeout elapses and as a result the Context Service terminates the activity, the transaction will also be terminated in the same state as the activity.

If the activity is instructed to complete in the Success state then the activity service sends an appropriate *completeWithStatus* message to the Tx ALS which will then try to commit. If there is only a single participant enrolled with the transaction then there is no need for the coordinator to execute the two-phase protocol. As such, the coordinator begins the *OnePhaseCommit* protocol and either transits to the *RolledBack* or *Committed* state, depending upon the result returned by the participant. The activity completion status is either *Failure* or *Success* respectively.

If there are multiple participants enrolled with the transaction, the coordinator transits to the *Preparing* state and begins to execute the two-phase commit protocol by sending the prepare message to each participant. If all of the participants indicate that the services they represent performed no work (i.e., are read only) then the transaction is complete and the coordinator transits to the *Committed* state; the activity completion status is *Success*.

Any failures from a participant or indication that it cannot prepare cause the coordinator to rollback (move to the *RollingBack* state) and send rollback messages to all of the other participants. It then transits to the *RolledBack* state, with the activity in the *Failure* completion status.

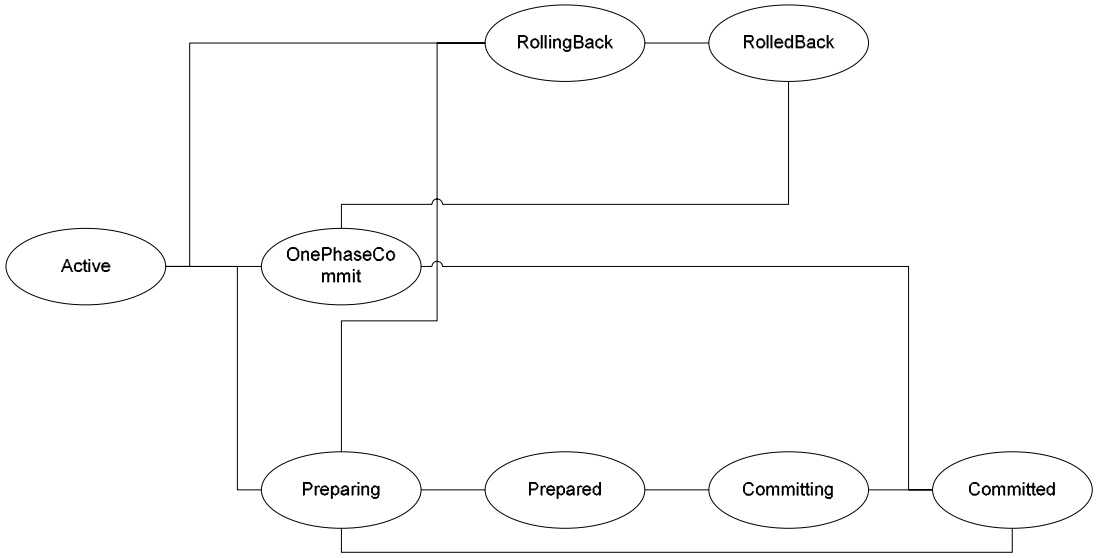

Figure 5, Transaction coordinator two-phase status transition.

Assuming all participants have prepared successfully, the transaction coordinator makes the decision as to whether to commit or rollback and must record sufficient information on stable storage to ensure this decision can be completed in the event of a failure. It is then in the *Prepared* state. When the coordinator starts the second phase of the commit protocol it is in the *Committing* state and ultimately moves to the *Committed* state.

In terms of the underlying activity service and coordination service, Figure 6 shows the flow of messages:

1) The application issues a *begin* on the Transaction/Coordination Service ALS to demarcate the beginning of a transaction. This causes the Transaction/Coordination Service to create a coordinator used to identify the activity instance and subsequently track the elements interested in the transaction outcome (i.e., the participants).

2) The application issues a server method. Context information is appended to the message. The context is used at the target to recreate the execution environment. Having been passed a coordinator reference the target registers interest in the transaction outcome. (Note, an implementation can choose to register participants directly to the coordinator or through a subordinate coordinator that resides on the target.)

3) The application issues a *completeWithStatus* to indicate the end of the transaction. The completion indication is passed to the coordinator. The coordinator sends the two-phase commit protocol messages to each registered participant and returns the outcome to the activity service.

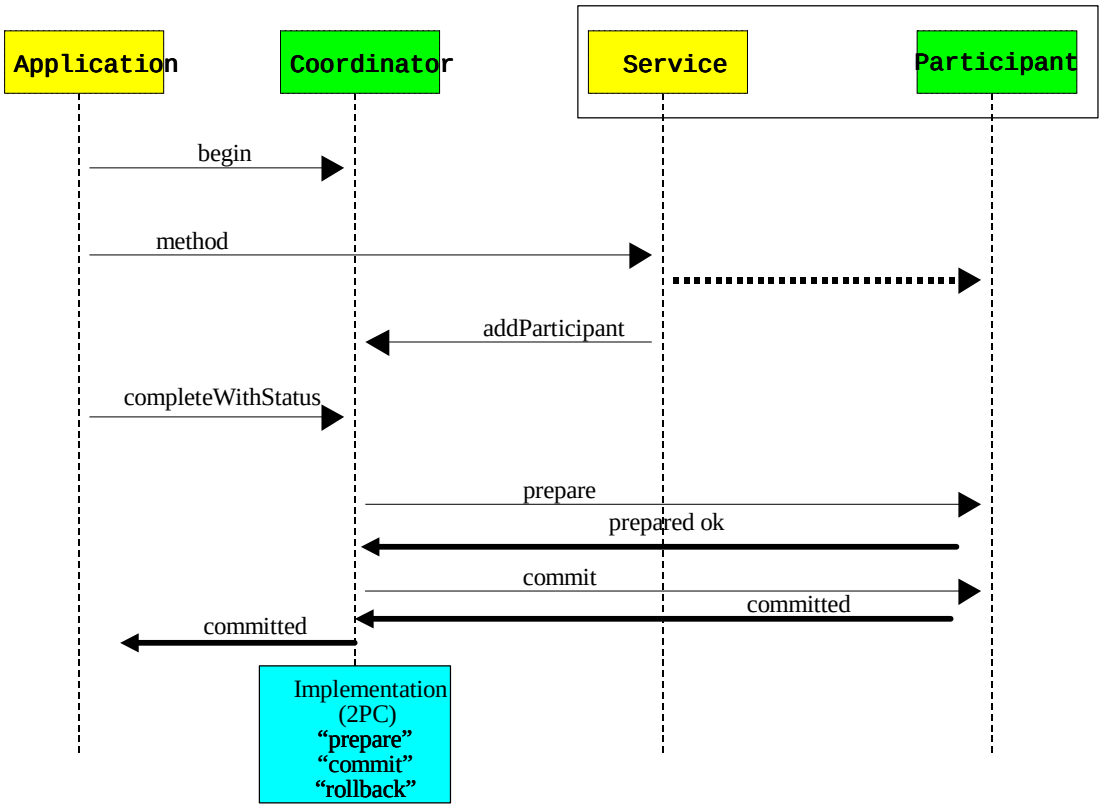

Figure 6, Two-phase commit transactions.

4.2.4 Two-phase participant state transitions

The participant state transitions are the same as the coordinators:

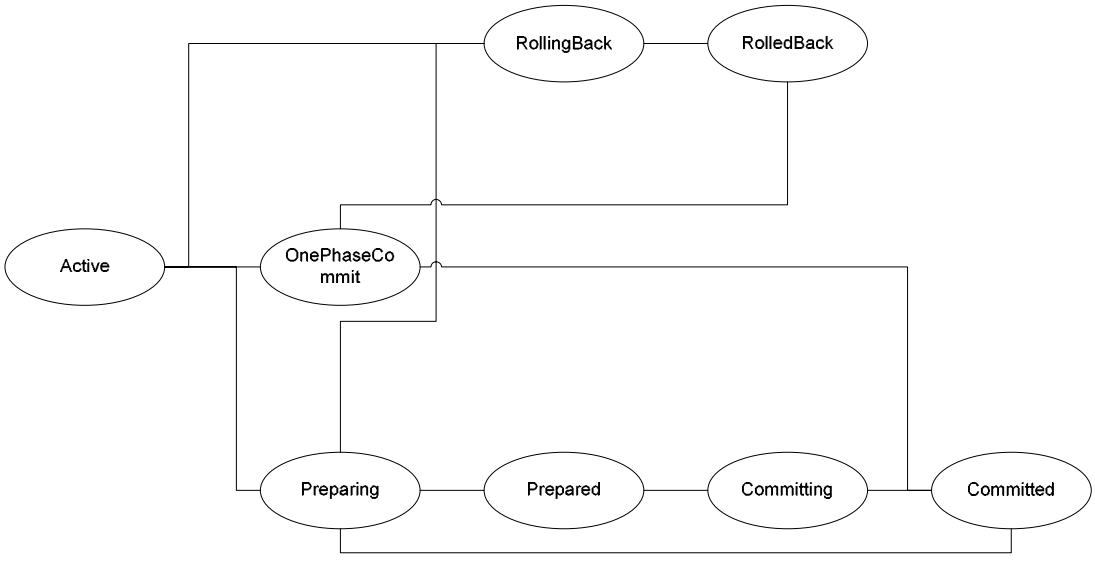

Figure 7, Two-phase participant state transitions.

4.2.5 Two-phase commit message interactions

In this section we shall describe the message that are exchanged between the coordinator and the participants. Although the text refers to the coordinator soliciting

responses from participants, because WS-CF supports an asynchronous model, participants may send unsolicited "responses" to the coordinator via the *setResponse* message.

The ACID transaction model supports two styles of participant service implementation: the *singleton* approach, whereby one participant service (end-point) is implicitly associated with only one transaction, and the *factory* approach, whereby a single participant service may manage participants on behalf of many different transactions. Therefore, all operations on the participant service are implicitly associated with the current context, i.e., it is propagated to the participants in order to identify which transaction is to be operated on. The unique participant identification is also present on each message.

The AT sub-protocol URI for the two-phase commit protocol is http://www.webservicestransactions.org/wsdl/wsTXM/tx-acid/2pc/2003/03 and this is used in the *addParticipant* invocation. The Participant accepts the following messages (illustrated in Figure 8). The CoordinatorParticipant end-point address as defined in WS-CF is propagated on all messages:

- prepare: The coordinator is preparing. The participant can respond with a *voteReadonly*, *voteCommit* or *voteRollback* messages indicating whether or not it is willing to commit. If *voteCommit* is used then optional Qualifiers may be sent back to augment the protocol. If the participant is a subordinate coordinator and finds that it cannot determine the status of some of its enlisted participants then it must return the *heuristicHazardFault* message. Alternatively, if a subordinate coordinator finds that some of the participants have committed and some have rolled back then it must return the *heuristicMixedFault* message.
- rollback: The coordinator is cancelling. If the participant is receiving this message after a *prepare* message, then any error at this point will cause a heuristic. If the participant is a subordinate coordinator and cannot determine how all of its enlisted participants terminated then it must return the *heuristicHazardFault* message. If the participant is a subordinate coordinator and some of its enlisted participants committed then it must return the *heuristicMixedFault* message. If the participant commits rather than rolls back then it must return the *heuristicCommitFault* message. Otherwise the participant sends the *rolledback* message.
- commit: The coordinator is top-level and is confirming. Any error at this point will cause a heuristic. If the participant is a subordinate coordinator and cannot determine how all of its enlisted participants terminated then it must return the *heuristicHazardFault* message. If the participant is a subordinate coordinator and some of its enlisted participants rolled back then it must return the *heuristicMixedFault* message. If the participant rolls back rather than commits then it must return the *heuristicRollbackFault* message. Otherwise the participant returns a *committed* message.

- onePhaseCommit: If only a single participant is registered with a two-phase coordinator then it is possible for the coordinator to optimize the commit stage and not have to execute two phases. If the participant is a subordinate coordinator and cannot determine how all of its enlisted participants terminated then it must return the *HeuristicHazardFault* message. If the participant is a subordinate coordinator and some of its enlisted participants rolled back then it must return the *HeuristicMixedFault* message. If the participant rolls back rather than commits then it must return the *HeuristicRollbackFault* message. Otherwise the participant returns either the *committed* or *rolledback* message.
- forgetHeuristic: The participant made a post-prepare choice that was contrary to the coordinator's. Hence it may have caused a non-atomic (heuristic) outcome. If this happens, the participant *must* remember the decision it took (persistently) until the coordinator tells it via this message that it is safe to forget. Success is indicated by sending the *heuristicForgotten* message. Any other response is assumed to indicate a failure.

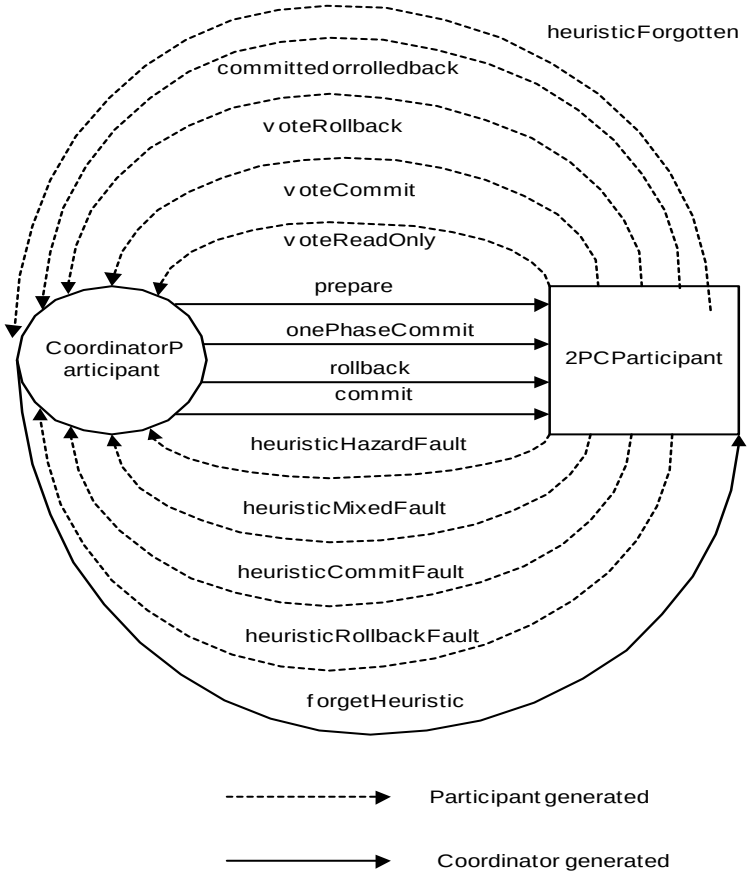

Figure 8, AT coordinator-to-participant message exchanges.

The WSDL portType declarations for the CoordinatorParticipant and twoPCParticipant roles are shown in Figure 9.

```
<wsdl:portType name="twoPCParticipantPortType">
  <wsdl:operation name="prepare">
```

```
    <wsdl:input message="tns:PrepareMessage"/>
  </wsdl:operation>
  <wsdl:operation name="onePhaseCommit">
    <wsdl:input message="tns:OnePhaseCommitMessage"/>
  </wsdl:operation>
  <wsdl:operation name="rollback">
    <wsdl:input message="tns:RollbackMessage"/>
  </wsdl:operation>
  <wsdl:operation name="commit">
    <wsdl:input message="tns:CommitMessage"/>
  </wsdl:operation>
  <wsdl:operation name="forgetHeuristic">
    <wsdl:input message="tns:ForgetHeuristicMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="CoordinatorParticipantPortType">
  <wsdl:operation name="committed">
    <wsdl:input message="tns:CommittedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="rolledBack">
    <wsdl:input message="tns:RolledBackMessage"/>
  </wsdl:operation>
  <wsdl:operation name="vote">
    <wsdl:input message="tns:VoteMessage"/>
  </wsdl:operation>
  <wsdl:operation name="heuristicForgotten">
    <wsdl:input message="tns:HeuristicForgottenMessage"/>
  </wsdl:operation>
  <wsdl:operation name="heuristicFault">
    <wsdl:input message="tns:HeuristicFaultMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

Figure 9, WSDL portType Declarations for Coordinator and 2PCParticipant Roles

Note, although an application Web Service may play the role of a participant, it is not required to.

4.2.6 Pre- and post- two-phase commit processing

Most modern transaction processing systems allow the creation of participants that do not take part in the two-phase commit protocol, but are informed before it begins and after it has completed. They are called Synchronizations, and are typically employed to flush volatile (cached) state, which may be being used to improve performance of an application, to a recoverable object or database prior to the transaction committing; once flushed, the data will the be controlled by a two-phase aware participant.

The AT sub-protocol URI for the synchronization protocol is http://www.webservicestransactions.org/wsdl/wsTXM/tx-acid/sync/2003/03 and this is used in the *addParticipant* invocation.

The message exchanges (ignoring the normal WS-CF coordinator-to-participant message exchanges, including failures) are illustrated in Figure 10:

- beforeCompletion: A Synchronization participant is informed that the coordinator it is registered with is about to complete the two-phase protocol and in what state, i.e., committing or rolling back. The failure of the participant at this stage will cause the coordinator to cancel if it is not already doing so.
- afterCompletion: A Synchronization participant is informed that the coordinator it is registered with has completed the two-phase protocol and in what state, i.e., committed or rolled back (via the associated Status). Any failures by the participant at this stage have no affect on the transaction.

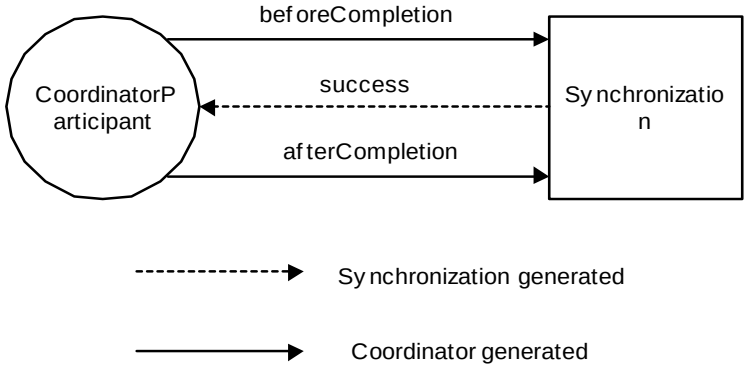

Figure 10, AT coordinator-to-synchronization message exchanges.

The WSDL portType declarations for the CoordinatorParticipant and Synchronization roles are shown in Figure 11.

```
<wsdl:portType name="SynchronizationPortType">
  <wsdl:operation name="beforeCompletion">
    <wsdl:input message="tns:BeforeCompletionMessage"/>
  </wsdl:operation>
```

```
  <wsdl:operation name="afterCompletion">
    <wsdl:input message="tns:AfterCompletionMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="CoordinatorParticipantPortType">
  <wsdl:operation name="beforeCompletionParticipantRegistered">
    <wsdl:input
message="tns:BeforeCompletionParticipantRegisteredMessage"/>
  </wsdl:operation>
  <wsdl:operation name="afterCompletionParticipantRegistered">
    <wsdl:input
message="tns:AfterCompletionParticipantRegisteredMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

Figure 11, WSDL portType Declarations for Coordinator and 2PCParticipant Roles.

Note, the participant is registered for both beforeCompletion and afterCompletion.

4.2.7 Coordinator state transitions for synchronization protocol

The state transitions for the transaction coordinator which has enrolled Synchronizations is shown in Figure 12. In this scenario we assume the transaction is committing: if it were to rollback, then only the *AfterCompletion* message will be sent from the coordinator to the Synchronization participants.

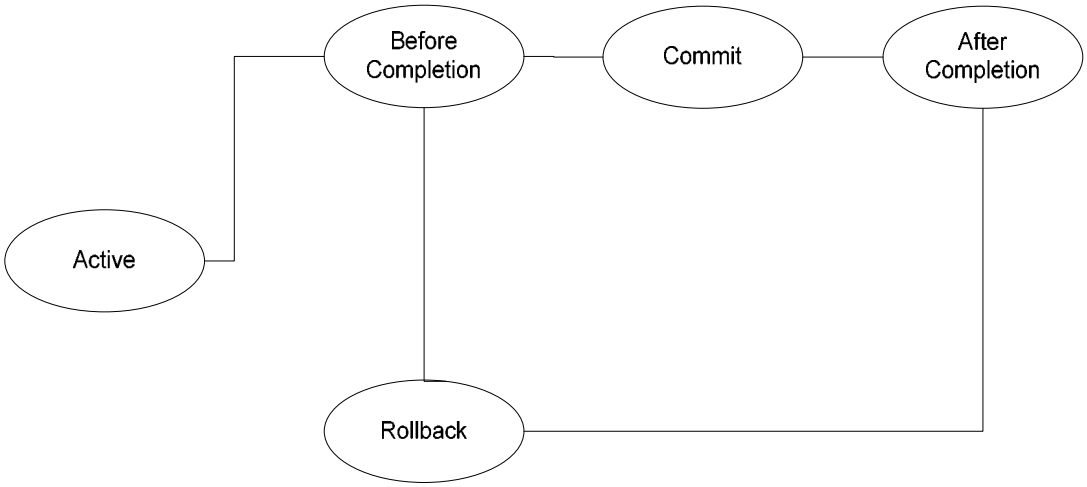

Figure 12, Transaction coordinator Synchronization state transitions.

The coordinator moves into the *BeforeCompletion* state and sends each enrolled Synchronization the *beforeCompletion* message. Any error received by the coordinator from a Synchronization at this stage will force the transaction to rollback. Assuming no errors occur, the two-phase commit protocol is executed, as detailed previously. Once the protocol has completed, the coordinator transits to the *AfterCompletion* status and sends the *afterCompletion* message to all Synchronizations; any errors at this stage do not affect the transaction outcome and how they are dealt with is implementation dependant.

4.2.8 Recovery and interposition

Because WS-TXM layers on WS-CF, interposition is allowed though not required. Individual participants may be subordinate coordinators to improve performance or to federate a distributed environment into separate domains (possibly managed by different organizations or transaction management systems).

Each participant or subordinate coordinator is responsible for ensuring that sufficient data is made durable in order to complete the transaction in the event of failures. *Recovering participants or coordinators use the recovery mechanisms defined in WS-CF to determine the current status of a transaction/participant and act accordingly.* Interposition and check pointing of state allow the system to drive a consistent view of the outcome and recovery actions taken, but allowing always the possibility that recovery isn't possible and must be logged or flagged for the administrator.

Although enterprise transaction systems address the aspects of distributed recovery, in a large scale environment or in the presence of long term failures, recovery may not be automatic. As such, manual intervention may be necessary to restore an application's consistency.

4.2.9 The context

```
<xs:complexType name="ContextType">
  <xs:complexContent>
    <xs:extension base="wstxm:ContextType"/>
  </xs:complexContent>
</xs:complexType>
<xs:element name="context" type="tns:ContextType"/>
```

Figure 13, Transaction Context.

4.2.10 Statuses

The following extensions to the WS-CTX Status type are returned by participants to indicate the outcome of executing relevant parts of the protocol and are also used to indicate the current status of the transaction:

- RollbackOnly: the status of the coordinator or participant is that it will rollback eventually.
- RollingBack: the coordinator or participant is in the process of rolling back.
- RolledBack: the coordinator/participant has rolled back. This may be a transient and in fact, because the protocol uses a presumed-abort optimisation, the NoActivity status can be used to infer that the coordinator cancelled.
- Committing: the coordinator/participant is in the process of committing. This does not mean that the final outcome will be Committed.
- Committed: the coordinator/participant has confirmed.
- HeuristicRollback: all of the participants rolled back when they were asked to commit.
- HeuristicCommit: all of the participants committed when they were asked to rollback.
- HeuristicHazard: some of the participants rolled back, some committed and the outcome of others is indeterminate.
- HeuristicMixed: some of the participants rolled back whereas the remainder committed.
- Preparing: the coordinator/participant is preparing.
- Prepared: the coordinator/participant has prepared.

These are specified in the AT schema, as per Figure 14..

```
<xs:simpleType name="StatusType">
  <xs:restriction base="wstxm:StatusType">
    <xs:enumeration value="activity.status.tx-acid.ROLLBACK_ONLY"/>
    <xs:enumeration value="activity.status.tx-acid.ROLLING_BACK"/>
    <xs:enumeration value="activity.status.tx-acid.ROLLED_BACK"/>
    <xs:enumeration value="activity.status.tx-acid.COMMITTING"/>
    <xs:enumeration value="activity.status.tx-acid.COMMITTED"/>
    <xs:enumeration value="activity.status.tx-
acid.HEURISTIC_ROLLBACK"/>
    <xs:enumeration value="activity.status.tx-
acid.HEURISTIC_HAZARD"/>
    <xs:enumeration value="activity.status.tx-acid.HEURISTIC_MIXED"/>
    <xs:enumeration value="activity.status.tx-acid.PREPARING"/>
    <xs:enumeration value="activity.status.tx-acid.PREPARED"/>
  </xs:restriction>
</xs:simpleType>
```

Figure 14, AT StatusType.

4.3 Long running action

The long running action model (LRA) is designed specifically for those business interactions that occur over a long duration. Within this model, an activity reflects business interactions: all work performed within the scope of an activity is required to be compensatable. Therefore, an activity's work is either performed successfully or undone. How services perform their work and ensure it can be undone if compensation is required, are implementation choices and not exposed to the LRA model. The LRA model simply defines the triggers for compensation actions and the conditions under which those triggers are executed.

As with most transaction models, LRA is concerned only with ensuring participants obey the protocol necessary to make an activity compensatable; semantics of the business interactions are not part of LRA model. Issues such as isolation of services between potentially conflicting activities and durability of service work are assumed to be implementation decisions. The coordination protocol used to ensure an activity is completed successfully or compensated is not two-phase and is intended to better model business-to-business interactions. Although this *may* result in non-atomic behaviour for the overall business activity, other activities may be started by the application or service to attempt to compensate in some other manner.

Each LRA is tied to the scope of an activity. This means that when the activity terminates, the LRA coordination protocol will be automatically performed either to accept or compensate the work.

In the LRA model, each activity is bound to the scope of a compensation interaction. For example, when a user reserves a seat on a flight, the airline reservation centre may take an optimistic approach and actually book the seat and debit the users account, relying on the fact that most of their customers who reserve seats later book them; the compensation action for this activity would obviously be to un-book the seat and credit the user's account. Work performed within the scope of a nested LRA must remain compensatable until an enclosing activity informs the service(s) that it is no longer required. For example, consider the night-out reservation example mentioned earlier.

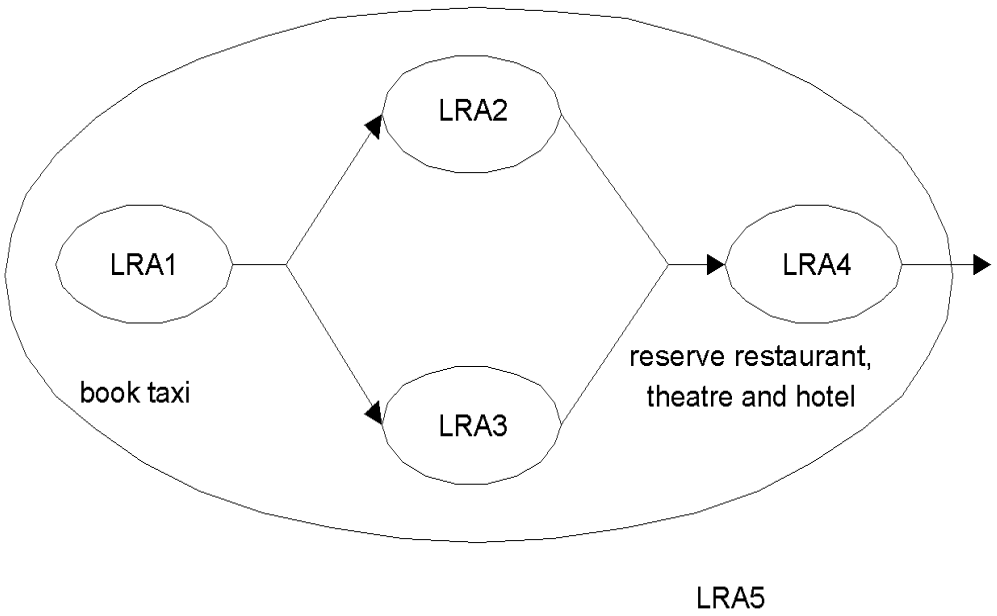

Figure 15, LRA example.

Figure 15 shows how part of the night-out may be mapped into LRAs. All of the individual activities are compensatable. For example, this means that if LRA1 fails or the user decides to not accept the booked taxi, the work will be undone automatically. Because LRA1 is nested within another LRA, once LRA1 completes successfully any compensation mechanisms for its work may be passed to LRA5: this is an implementation choice for the Compensator. In the event that LRA5 completes successfully, no work is required to be compensated, otherwise all work performed within the scope of LRA5 (LRA1 to LRA4) will be compensated.

The coordinator-type URI for the LRA model is http://www.webservicestransactions.org /wstxm/tx-lra/2003/03

In the following sections we shall use the terms LRA and compensation activity interchangeably.

4.3.1 Restrictions imposed on using WS-CF

As well as the restrictions outlined previously for general WS-TXM protocols, the LRA transaction model imposes the following additional restrictions:

- It is illegal to call the WS-CF *coordinate* operation on the coordinator. All WS-TXM coordinator implementations will return the *notCoordinated* message if the coordinator receives a *coordinate* request.

4.3.2 Context

The context for the LRA protocol, from the LRA schema, is shown in Figure 16.

```
<xs:element name="context">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="wstxm:ContextType">
        <xs:sequence>
          <xs:element name="lra-id" type="xs:anyURI"/>
          <xs:element name="coordinator-hierarchy">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="coordinator-location"
                  type="xs:anyURI" minOccurs="0"
                  maxOccurs="unbounded"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Figure 16, The LRA Protocol Context.

4.3.3 Services and Compensators

As in any business interaction, application services may or may not be compensatable. Even the ability to compensate may be a transient capability of a service. An ALS for the LRA is associated with the underlying CTX Service. It is up to the user of the augmented context as to whether or not all services invoked within the scope of an LRA must be compensatable. This choice is made by setting the MustUnderstand attribute appropriately. Obviously by mixing the two service types the user may end up with a business activity that will ultimately not be undone by the LRA model, but which may require outside (application specific) compensation.

A Compensator is the LRA participant that operates on behalf of a service to undo the work it performs within the scope of an LRA or to compensate for the fact that the original work could not be completed. How compensation is carried out will obviously be dependant upon the service; compensation work may be carried out by other LRAs which themselves have Compensators.

For example, consider the travel example illustrated in Figure 17, where normal (non-failure) activities are connected by solid lines whereas compensation activities are connected by dashed lines. In this case the user first attempts to book a first-class seat on an airline; the compensator for this (which is executed in the event of a crash or failure to complete the booking, for example) starts another LRA that tries to cancel the booking. If the cancellation LRA fails, then its compensator emails the system administrator for the airline reservation site; if the cancellation succeeds, however, it tries to book an economy seat on the same flight (which for simplicity does not have a compensator task).

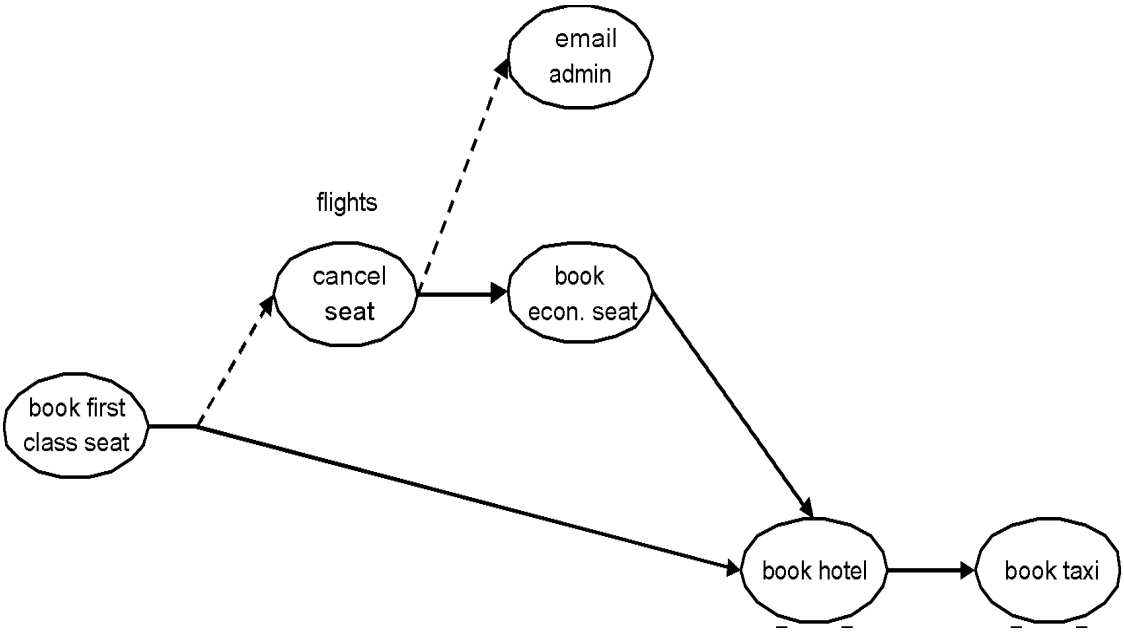

Figure 17, Compensator LRAs.

Because LRAs may execute over a long period of time, compensation may have to occur at any time and be tolerant of failures. Consequently, Compensators may have to maintain information within a durable form.

When a service does work that may have to be later compensated within the scope of an LRA, it enlists a Compensator participant with the LRA coordinator. The Compensator (which is a WS-CF Participant), will be invoked in the following way (illustrated in Figure 18) by the LRA coordinator when the activity terminates:

- Success: the activity has completed successfully. If the activity is nested then Compensators may propagate themselves (or new Compensators) to the enclosing LRA. Otherwise the Compensators are informed that the activity has terminated and they can perform any necessary cleanups.

- Fail: the activity has completed unsuccessfully. All Compensators that are registered with the LRA will be invoked to perform compensation in the reverse order. The coordinator forgets about all Compensators that indicated they operated correctly. Otherwise, compensation may be attempted again (possibly after a period of time) or alternatively a compensation violation has occurred and must be logged.

Each service is required to log sufficient information in order to ensure (with best effort) that compensation is possible. Because WS-TXM layers on WS-CF, interposition is allowed though not required. Individual compensators may be subordinate coordinators to improve performance or to federate a distributed environment into separate domains.

Each compensator (participant) or subordinate coordinator is responsible for ensuring that sufficient data is made durable in order to undo the LRA in the event of failures. Recovering participants or coordinators use the recovery mechanisms defined in WS-CF to determine the current status of the LRA and act accordingly. Interposition and check pointing of state allow the system to drive a consistent view of the outcome and recovery actions taken, but allowing always the possibility that recovery isn't possible and must be logged or flagged for the administrator. In a large scale environment or in the presence of long term failures, recovery may not be automatic. As such, manual intervention may be necessary to restore an application's consistency.

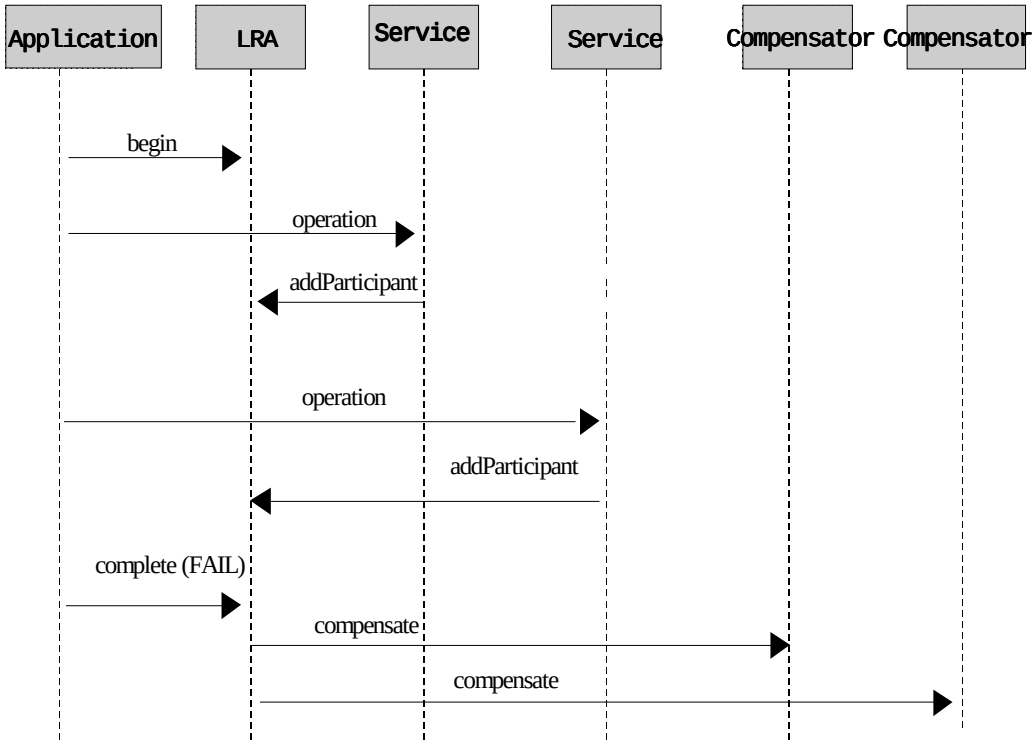

Figure 18, Example LRA interaction diagram.

In order to conduct the protocol, the following message interactions occur between the coordinator and its participants (Compensators):

- Compensator: this accepts the *compensate* and *complete* messages, indicating it should either compensate for work or tidy up respectively. A Compensator that cannot compensate must maintain its information until it is told to *forget*.
- Coordinator: the responses the coordinator accepts are *compensated*, *unknownCompensatorFault*, *cannotCompensateFault*, *completed*, *forgot* and *cannotCompleteFault*.

The message interactions (via the normal WS-CF Coordinator/Participant exchanges) are shown in Figure 19; not shown are the other Coordinator/Participant message exchanges.

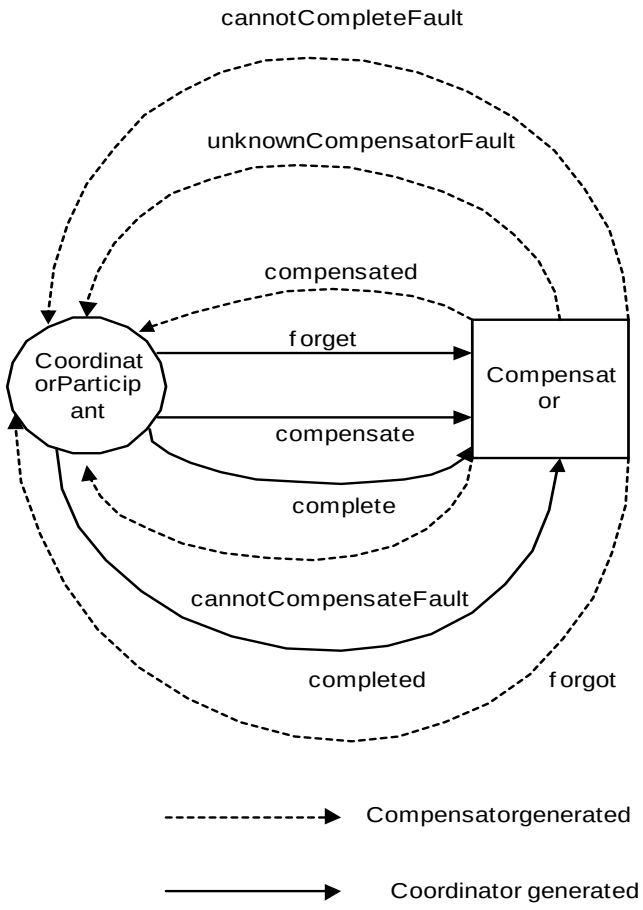

Figure 19, Coordinator-to-Compensator message interactions.

It is expected that the receipt of *cannotCompensateFault* or *cannotCompleteFault* will be handled by the application or logged if not.

The WSDL portType declarations for the CoordinatorParticipant and Compensator roles are shown in Figure 20.

```
<wsdl:portType name="CompensatorPortType">
  <wsdl:operation name="compensate">
    <wsdl:input message="tns:CompensateMessage"/>
  </wsdl:operation>
```

```
  <wsdl:operation name="complete">
    <wsdl:input message="tns:CompleteMessage"/>
  </wsdl:operation>
  <wsdl:operation name="forget">
    <wsdl:input message="tns:ForgetMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="CoordinatorPortType">
  <wsdl:operation name="compensated">
    <wsdl:input message="tns:CompensatedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="completed">
    <wsdl:input message="tns:CompletedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="forgot">
    <wsdl:input message="tns:ForgotMessage"/>
  </wsdl:operation>
  <wsdl:operation name="unknownCompensator">
    <wsdl:input message="tns:UnknownCompensatorFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="cannotCompensate">
    <wsdl:input message="tns:CannotCompensateFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="cannotComplete">
    <wsdl:input message="tns:CannotCompleteFaultMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

Figure 20, WSDL portType Declarations for CoordinatorParticipant and Compensator Roles.

Note, a Compensator can resign from the LRA at any time prior to the completion of an activity by sending the *removeParticipant* message to the coordinator.

4.3.4 Qualifiers

When a Compensator is enrolled with an LRA, the entity performing the enrol can supply a number of qualifiers which may be used by the coordinator and business

application to influence the overall outcome of the activity. The currently supported qualifiers are:

- TimeLimit: the time limit (in seconds) that the Compensator can guarantee that it can compensate the work performed by the service. After this time period has elapsed, it may no longer be possible to undo the work within the scope of this (or any enclosing) LRA. It may therefore be necessary for the application or service to start other activities to explicitly try to compensate this work. The application or coordinator may use this information to control the lifecycle of an LRA.

The time limit qualifier from the LRA schema is shown in Figure 21.

```
<xs:element name="TimeLimitQualifier">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="wstxm:TimeLimitQualifierType">
        <xs:sequence>
          <xs:element name="compensator" type="xs:anyURI"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Figure 21, The Time Limit Qualifier.

4.3.5 Coordinator

The LRA model uses a *presumed nothing protocol*: the coordinator must communicate with Compensators in order to inform them of the LRA activity. Every time a Compensator is enrolled with an LRA, the coordinator must make information about it durable so that the Compensator can be contacted when the LRA terminates, even in the event of subsequent failures.

4.3.6 Independent LRAs and application structuring

So far we have not really considered the relationship between LRAs in an application. Obviously LRAs may be used sequentially and concurrently as illustrated in Figure 15, where the termination of an LRA signals the start of some other unit of work within an application. However, LRAs are units of compensatable work and an application may have as many such units of work operating simultaneously as it needs to accomplish its tasks. Furthermore, the outcome of work within LRAs may determine how other LRAs are terminated.

An application can be structured to so that LRAs are used to assemble units of compensatable work and then held in the active state while the application performs other work in the scope of different (concurrent or sequential) LRAs. Only when the right subset of work (LRAs) is arrived at by the application will that subset be confirmed; all other LRAs will be told to cancel (complete in a failure state).

For example, Figure 18 illustrates how a travel agency application may be structured using this technique. LRA1 is used to obtain the taxi to the airport. The user then wishes to get the cheapest flight from three different airlines and so the agency structures each seat reservation (or booking, depending upon how the user feels) as a separate LRA. In this example, the airline represented by LRA2 gives a cost of $150 for the flight; while LRA2 is still active, the application starts LRA3, a new independent-level LRA to ask the next airline for a costing: LRA3 gives a value of $160 and so it is cancelled. Finally the travel agency starts LRA4 to check the other airline, which gives a value of $120 for the seat. Thus, LRA2 is cancelled and LRA4 is confirmed, with the result that the seat is bought. The travel agency then uses the same technique to select the cheapest travel insurance from amongst two options (using LRA5 and LRA6).

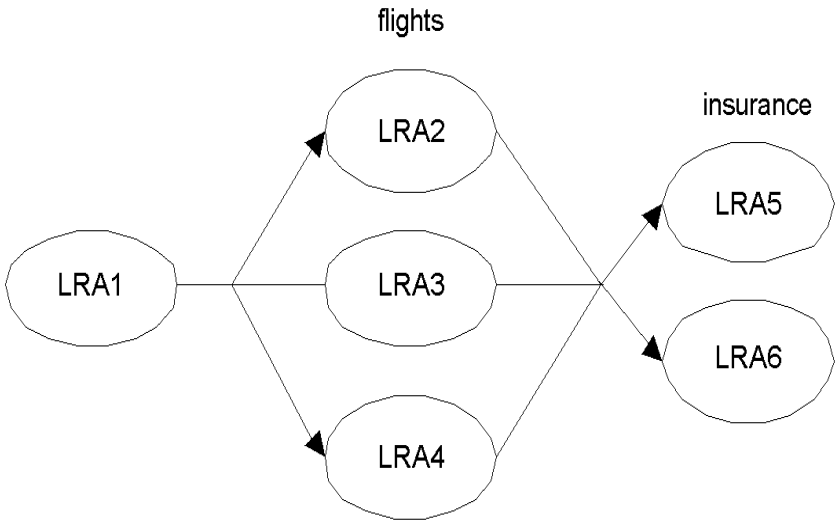

Figure 22, Using LRAs to select units of work.

4.3.7 Status values

The following extensions to the WS-CTX Status type are returned by Compensators to indicate the current status:

- Compensating: the Compensator is currently compensating for the LRA.
- Compensated: the Compensator has successfully compensated for the LRA.
- FailedToCompensate: the Compensator was not able to compensate for the LRA. It must maintain information about the work it was to compensate until the coordinator sends it a *forget* message.
- Completing: the Compensator is tidying up after being told to complete.
- Completed: the coordinator/participant has confirmed.

- FailedToComplete: the Compensator was unable to tidy-up.

4.4 Business process transaction

In the business process transaction model (BP model) all parties involved in a business process reside within *business domains*, which may themselves use business processes to perform work. Business process transactions are responsible for managing interactions *between* these domains. A business process (business-to-business interaction) is split into *business tasks* and each task executes within a specific business domain. A business domain may itself be subdivided into other business domains (business processes) in a recursive manner. An individual task may require multiple services to work. Each task is assumed to be a compensatable unit of work. However, as with the LRA model described earlier, how compensation is provided is an implementation choice for the task.

For example, consider the purchasing of a home entertainment system example shown in Figure 23. The on-line shop interacts with its specific suppliers, each of which resides in its own business domain. The work necessary to obtain each component is modelled as a separate task. In this example, the HiFi task is actually composed of two sub-tasks.

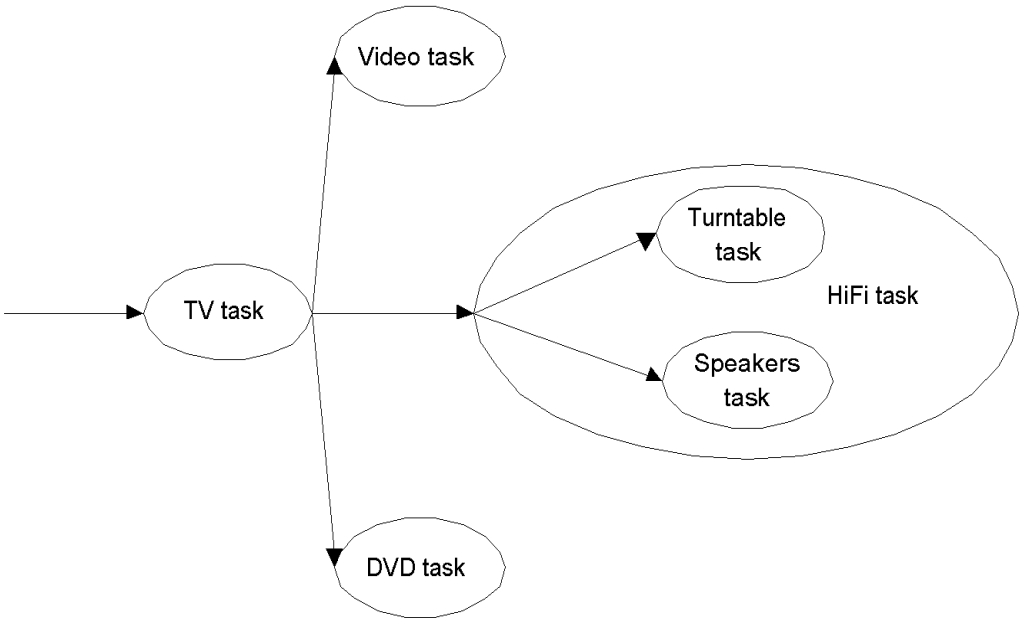

Figure 23, Business processes and tasks.

In this example, the user may interact synchronously with the shop to build up the entertainment system. Alternatively, the user may submit an order (possibly with a list of alternate requirements) to the shop which will eventually call back when it has been filled; likewise, the shop then submits orders to each supplier, requiring them to call back when each component is available (or is known to be unavailable).

The business process transaction model supports this synchronous and asynchronous interaction pattern. Business domains are instructed to perform work within the scope of a global business process. The business process has an overall manager that may be informed by individual tasks when they have completed their work (either

successfully or unsuccessfully), or it may periodically communicate with each task to determine its current status. In addition, each task may make period checkpoints of its progress such that if a failure occurs, it may be restarted from that point rather than having to start from the beginning. A business process can either terminate in a confirmed (successful) manner in which case *all* of the work requested will have been performed, or it will terminate in a cancelled (unsuccessful) manner, in which case all of the work will be undone.

If it cannot be undone, then this fact must be logged. One key difference between the business process transaction model and that of traditional 2PC is that it assumes success, that is the BP model is optimistic and assumes the failure case is the minority and can be handled or resolved offline if necessary, or through replay/void/compensation, but not always automatically, often requiring human interaction.

Just as this specification does not mandate how individual business tasks perform work, it does not mandate how a business task undoes its work. It may use a compensation mechanism similar to the LRA model presented earlier, or it may use some other framework.

Logging is essential in the BP model for replay, void, and compensation (attempts to "rollback" or restore the initial state). However, recovery may ultimately be the responsibility of a manual operator if automatic recovery/compensation is not possible. As we shall see, interposition plays a major role in the BP model to improve performance or to federate a distributed environment into separate domains. In fact user intervention is likely to be an important part of business process management, as is monitoring of every step. As such, although the protocols are described in terms of participant services, it is expected that in some cases implementations of participants will interact directly with operators. The asynchronous nature of the BP model allows arbitrary periods of time to elapse between requests and responses to assist in this type of interaction.

Each participant or subordinate coordinator is responsible for ensuring that sufficient data is made durable in order to complete the BP even in the event of failures. Recovering participants or coordinators use the recovery mechanisms defined in WS-CF to determine the current status of the BP and act accordingly. Interposition and check pointing of state allow the system to drive a consistent view of the outcome and recovery actions taken, but allowing always the possibility that recovery isn't possible and must be logged or flagged for the administrator. In a large scale environment or in the presence of long term failures, recovery may not be automatic. As such, manual intervention may be necessary to restore an application's consistency.

A business process transaction is associated with an activity, such that the lifetime of the activity is the lifetime of the business process (essentially the lifetime of the business-to-business interaction). The coordinator-type URI for the BP model is http://www.webservicestransactions.org/wstxm/tx-bp/2003/03

Figure 24 illustrates the state transitions for a business process (and business task).Once created, the business process (which is structured as an activity) is in the *Active* state. From here it may transit to the *Cancelled* state and in which case no

further work is performed. More typical is that it moves to the *Working* state where it may remain for as long as is necessary to perform the work necessary.

How (and where to) the business process moves from this state will depend upon the application and the structure of any individual business tasks. If there are no failures (e.g., all work requested can be performed) then the process moves to the *Confirmed* state and all work is completed. However, if there are failures (e.g., a machine crash or the fact that a requested item cannot be found to fulfil an order) then the process may either move to the cancelled state (signifying that all work performed has been undone) or it moves to the *Failure* state where business-level compensation (or other recovery mechanisms such as void and replay) may occur.

This compensation is different from that which occurs to undo the entire business process: it is an attempt by each task/process to compensate for the inability to fulfil a specific business requirement. If it is possible to compensate then the task moves back to the *Working* state; otherwise it moves to the *Cancelled* state. Because compensation may occur in an application/domain specific manner it may include manual (operator) involvement. As such, compensation can take arbitrary amounts of time.

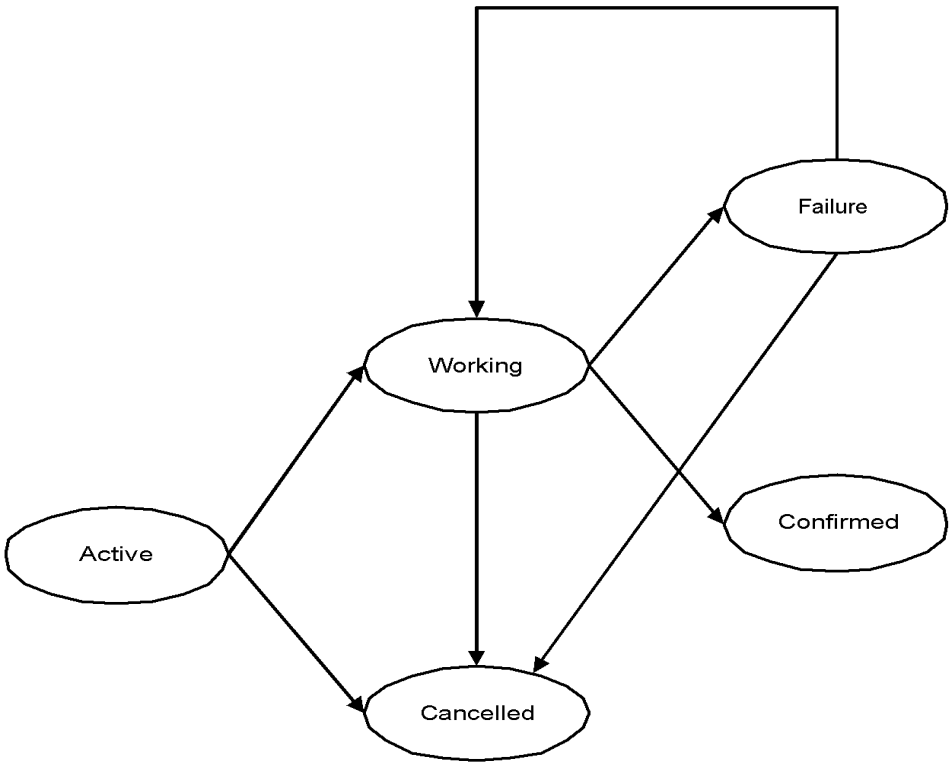

Figure 24, Business process state transitions.

4.4.1 Context

The context type for the Business Process sub-protocols is shown in

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

```
      <xs:sequence>
        <xs:element name="process-id" type="xs:anyURI"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 25, The BP Protocol Context Type

4.4.2 Business domains and interposition

In order to participant within a business process transaction (BP model), each business domain is exposed as a single subordinate (interposed) coordinator, forming a parent-child relationship; the business domain is identified by the coordinator URI. The interposed coordinator is responsible for managing the domains participation within the overall business transaction. The internal implementation of a domain is not the responsibility of this specification. In order to perform work necessary for a business task a domain may use its own business process transaction, ACID transactions, or some other infrastructure.

For example, Figure 26 shows how the home entertainment system would be federated into interposed coordinator domains. Each domain is represented by a subordinate coordinator that masks the internal business process infrastructure from its parent. Not only does the interposed domain require the use of a different context when communicating with services within the domain (the coordinator endpoint is different), but each domain may use different protocols to those outside of the domain: the subordinate coordinator may then act as a translator from protocols outside the domain to protocols used within the domain.

For example, a domain may be implemented entirely using the OASIS BTP with the interposed coordinator responsible for mapping BP protocol messages into BTP's atom or cohesion messages and vice versa. Another domain (possibly in the same overall business process) may use the OMG's Object Transaction Service (OTS) and therefore provide an interposed coordinator to translate between the BP model and the OTS. The important point is that as far as a parent coordinator in the BP hierarchy is concerned it interacts with participants and as long as those participants obey the BP protocol, it cannot determine the implementation.

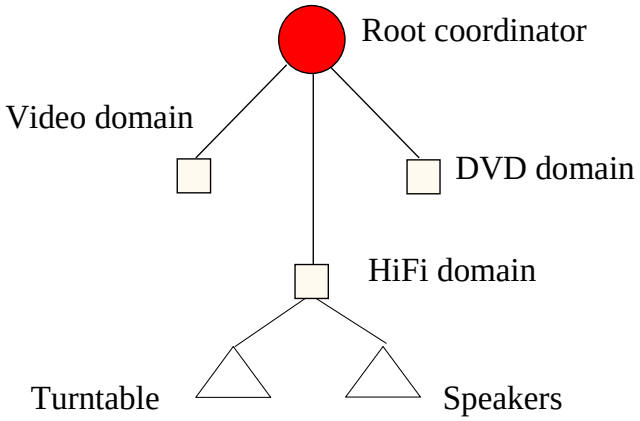

Figure 26, Example business process interposition.

This specification does not define what constitutes a business domain, i.e., what collection of services are grouped into a single domain. Neither does it specify how or when interposition occurs on behalf of a given domain. For example, the first service that resides within a specific domain may be responsible for performing interposition. Or interposition may occur only when work is done that is to be controlled by the overall business process.

When and where to use interposition is a design decision. Typically there will be a business transaction manager (e.g., the root coordinator) that knows which systems have been updated or not (i.e., knows which systems have completed and with what status). It knows either because the participants have notified it of the completion and status, or because it asked for a response and did not get one. Or it knows some parts have finished with others still pending.

Unlike traditional transaction systems where recovery is typically automatic and required little or no user or application intervention, in a long running business process that executes over many disparate business domains, the hardest part is determining when a failure has occurred and what to do in such a situation. In the worst case, the error is logged and brought to the attention of an administrator. In the best case, the compensation or undo logic is executed, or some workaround is found through presenting the exception to the user and having the user choose another possible option.

4.4.3 Protocols

The protocols used by the business process transaction model can be categorized as predominately driven *from* the business domain or *to* the business domain.

4.4.3.1 From the business domain

It is important to understand how a business process transaction is controlled and ultimately terminated. In a traditional transaction system, there is a single transaction *terminator* (the entity that ultimately informs the coordinator to commit or rollback). However, in a business process transaction there may never be a single terminator or the role of the terminator may flow with the business interactions. Each entity that wishes to be informed when the business process can be terminated may enlist a

TerminatorParticipant with the coordinator, which is an instance of the WS-CF Participant.

The TerminatorParticipant uses the *terminate-notification* sub-protocol, which is identified by the URI http://www.webservicestransactions.org/wstxm/tx-bp/tn/2003/03

This protocol is executed automatically by the coordinator when the business process enters a completion state. The completion state being defined by everything running to successful conclusion, cancel, compensation, or user workaround. The Participant accepts the following messages (illustrated in Figure 27):

- *confirmComplete*: the business process can be completed successfully. The TerminatorParticipant responds with the *confirmCompleted* message. Any other response is ignored by the coordinator.
- *cancelComplete*: the business process can only be cancelled. The TerminatorParticipant calls back with the *cancelCompleted* message. Any other response is ignored by the coordinator.

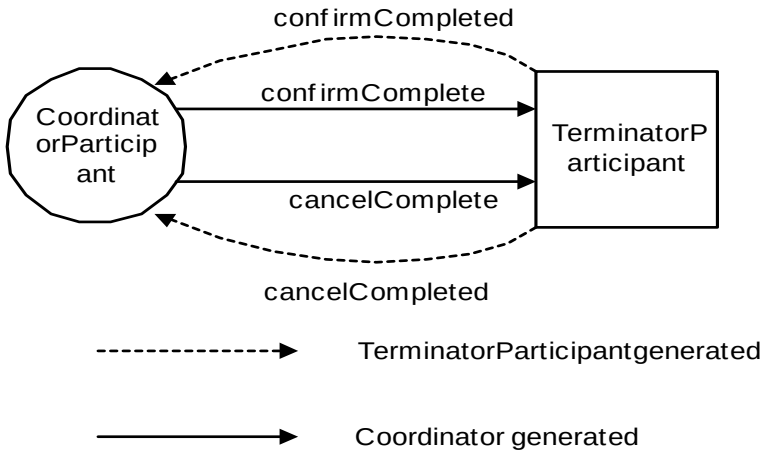

Figure 27, Coordinator-to-terminator interactions.

The WSDL portType declarations for the CoordinatorParticipant and TerminatorParticipant roles is shown in Figure 28.

```
<wsdl:portType name="TerminatorParticipantPortType">
  <wsdl:operation name="confirmComplete">
    <wsdl:input message="ConfirmCompleteMessage"/>
  </wsdl:operation>
  <wsdl:operation name="cancelComplete">
    <wsdl:input message="CancelCompleteMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="CoordinatorParticipantPortType">
```

```
  <wsdl:operation name="confirmCompleted">
    <wsdl:input message="ConfirmCompletedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="cancelCompleted">
    <wsdl:input message="CancelCompletedMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

Figure 28, The CoordinatorParticipant and TerminatorParticipant Roles

It is beyond the scope of this specification to determine whether or not only a single TerminatorParticipant can be enlisted with the business process, since this will depend upon the application.

If a business domain finds that it cannot fulfil its work it can either cause the entire business process to fail (set the CompletionStatus to FAIL_ONLY), or it can give its parent the opportunity to perform some compensation. This compensation will be specific to the business process, but could include removing the failed business domain from the overall process or asking the domain to perform some alternate work. In this case, the *parent* registers a *BusinessProcessParticipant* with the *child* domain (interposed coordinator) and uses the *businessProcess* protocol.

The BP model sub-protocol URI for the *businessProcess* protocol is http://www.webservicestransactions.org/wstxm/tx-bp/bp/2003/03 and this is used in the *addParticipant* invocation. The *businessProcess* protocol can be executed as many times as necessary within the business process by sending the appropriate WS-CF *coordinate* message to the coordinator.

The Participant accepts the following messages (illustrated in Figure 24); the CoordinatorParticipant end-point address as defined in WS-CF is propagated on all messages:

- *failure*: the child (interposed) coordinator sends this message to the participant (and indirectly to its parent) to indicate that it cannot perform the requested work and has cancelled. Because the message is contextualized, the identification of the business domain (coordinator URI) is included in the message. The BusinessProcessParticipant sends the *failureAcknowledged* message. Any other response is ignored.
- *failureHazard*: the child (interposed) coordinator sends this message to the participant (and indirectly to its parent) to indicate that it cannot perform the requested work and has been unable to cancel completely. Because the message is contextualized, the identification of the business domain (coordinator URI) is included in the message. The BusinessProcessParticipant sends the *failureHazardAcknowledged* message. Any other response is ignored.

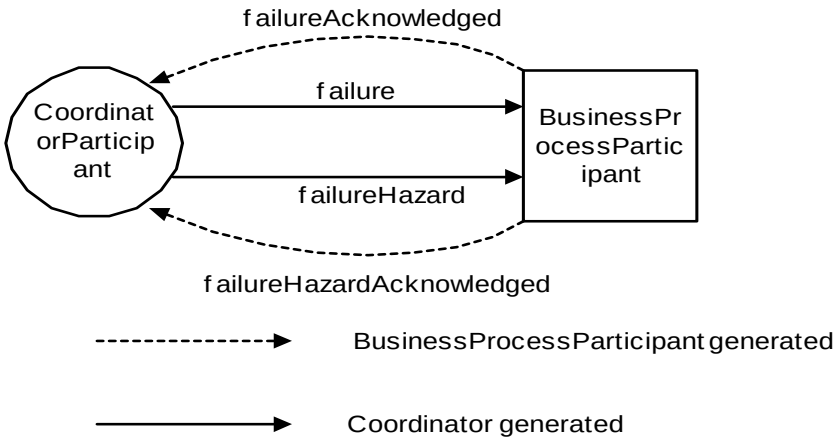

Figure 29, Business process protocol interactions.

The WSDL portType declarations for the CoordinatorParticipant and BusinessProcessParticipant roles are shown in

```
<wsdl:portType name="BusinessProcessParticipantPortType">
  <wsdl:operation name="failure">
    <wsdl:input message="FailureMessage"/>
  </wsdl:operation>
  <wsdl:operation name="failureHazard">
    <wsdl:input message="FailureHazardMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="CoordinatorParticipantPortType">
  <wsdl:operation name="failureAcknowledged">
    <wsdl:input message="FailureAcknowledgedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="failureHazardAcknowledged">
    <wsdl:input message="FailureHazardAcknowledgedMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

Figure 30, CoordinatorParticipant and BusinessProcessParticipant portType Declarations.

4.4.3.2 To the business domain

Within the BP model, each business domain is represented by a *BusinessTaskCoordinator*. The identifier of the BusinessTaskCoordinator may be used at the application (business process) level to determine task associations. Because a business process is modelled as an activity, a BusinessTaskCoordinator is implicitly bound to a single business process. The business domain is responsible for

managing the work given to it within the scope of the activity and mapping it to the business process, presumably via the activity identifier. The BP protocol defines the following coordination protocols for which a BusinessTaskCoordinator can enlist:

- *checkpoint:* when instructed to do so, each domain creates a checkpoint from which it can be restarted should it be instructed to later. Each checkpoint is uniquely identified and the identifier can later be used to restart the business process from a specific checkpoint. The failure (or inability) of a domain to create a checkpoint invalids the entire checkpoint. The *checkpointTimelimit* Qualifier may be returned to indicate for how long the checkpoint will be valid.
- *restart*: each domain is instructed to restart itself from the specified checkpoint.
- *workStatus*: determines whether or not the individual domains have completed their work.
- *completion*: called on all business domains to either confirm or cancel the work of the business process.

The BP sub-protocol URI for the *checkpoint* protocol is http://www.webservicestransactions.org/wstxm/tx-bp/cp/2003/03 and this is used in the *addParticipant* invocation. The *checkpoint* protocol can be executed as many times as necessary by the application within the business process by sending the appropriate WS-CF *coordinate* message to the coordinator.

The Participant accepts the following messages (illustrated in Figure 31).

- *createCheckpoint*: the business domain should attempt to create a consistent checkpoint and associate it with the unique identifier provided by the coordinator. The checkpoint is required to be maintained for the duration of the business process. If the business domain can create the checkpoint then it sends the *checkpointed* message to the CoordinatorParticipant. If the checkpoint cannot be created then the *checkpointFailed* message is sent. Failure to create a checkpoint does not automatically mean that the business process fails; as such the completion status of the activity is not changed.

If a consistent checkpoint is obtained across all business domains, the *checkpointingSucceeded* message is sent to the ClientRespondant (the endpoint representing the entity that sent the *coordinate* message) including the unique checkpoint identifier. Otherwise a *checkpoiningtFailed* message is sent.

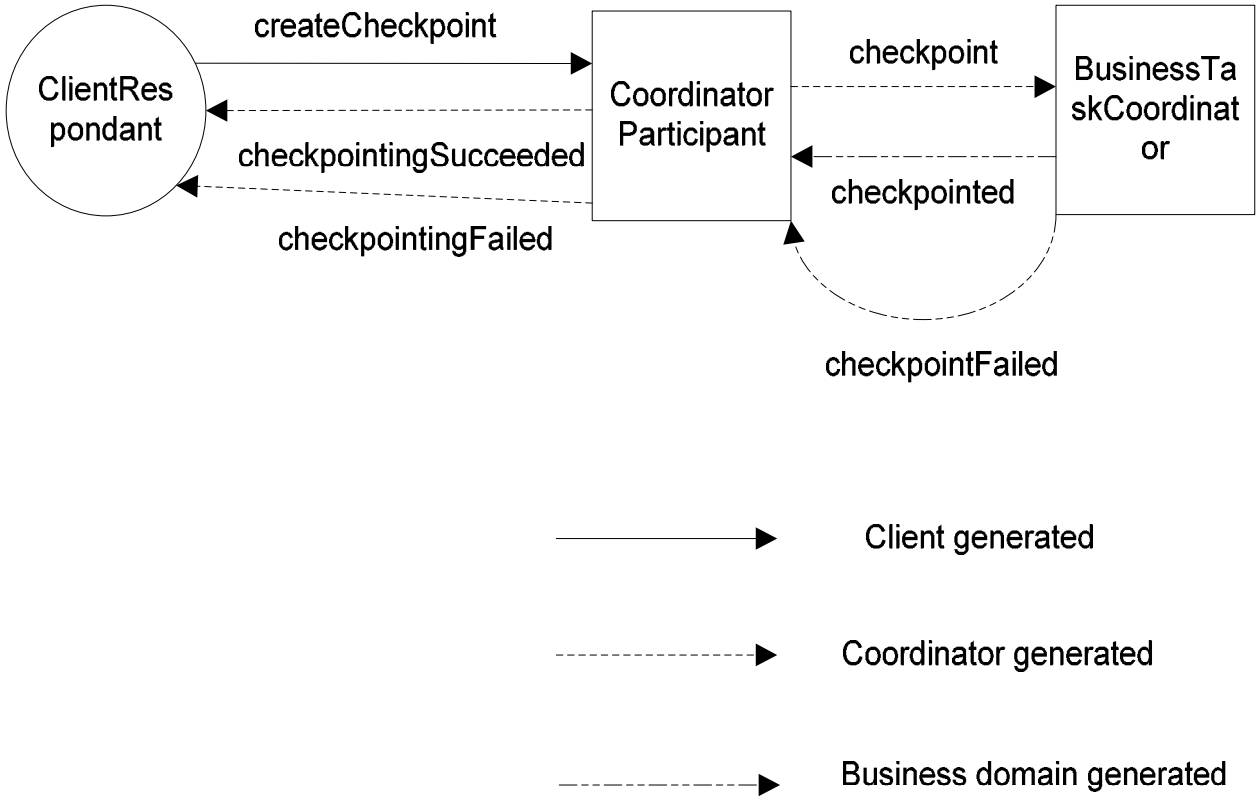

Figure 31, Checkpoint protocol interactions.

The actor definitions for the Checkpoint protocol are shown in Figure 31.

```
<wsdl:portType name="ClientRespondantPortType">
  <wsdl:operation name="checkpointingSuccedded">
    <wsdl:input message="tns:CheckpointingSucceededfulMessage"/>
  </wsdl:operation>
  <wsdl:operation name="checkpointingFailed">
    <wsdl:input message="tns:CheckpointingFailedMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="CoordinatorParticipantPortType">
  <wsdl:operation name="createCheckpoint">
    <wsdl:input message="tns:CreateCheckpointMessage"/>
  </wsdl:operation>
  <wsdl:operation name="checkpointed">
    <wsdl:input message="tns:CheckpointedMessage"/>
  </wsdl:operation>
```

```
  <wsdl:operation name="checkpointFailed">
    <wsdl:input message="tns:CheckpointFailedMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="BusinessTaskCoordinatorPortType">
  <wsdl:operation name="checkpoint">
    <wsdl:input message="tns:CheckpointMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

Figure 32, Checkpoint Protocol Actor WSDL portType Declarations.

The BP sub-protocol URI for the *restart* protocol is http://www.webservicestransactions.org/wstxm/tx-bp/restart/2003/03 and this is used in the *addParticipant* invocation. The *restart* protocol can be executed as many times as necessary by the application within the business process by sending the appropriate WS-CF *coordinate* message to the coordinator.

The Participant accepts the following messages (illustrated in Figure 33); the CoordinatorParticipant end-point address as defined in WS-CF is propagated on all messages:

- *restart*: the unique checkpoint identifier is included in the *tryRestart* message and instructs the business domain to restart its work from the specified checkpoint. Any work that may have occurred subsequent to this checkpoint must be undone by the domain. If the domain cannot restart from the specified checkpoint (e.g., it cannot undo additional work) then it sends the *cannotRestart* message to the CoordinatorParticipant. Otherwise the *restarted* message is sent.

If the checkpoint identifier is invalid, the coordinator sends the *invalidCheckpoint* message to the ClientRespondant ((the endpoint representing the entity that sent the *coordinate* message). If a consistent restart is obtained across *all* business domains, the *restartedSuccessfully* message is sent to the ClientRespondant including the unique checkpoint identifier. Otherwise a *restartFailed* message is sent and the completion status of the activity representing the business process is set to FAIL_ONLY.

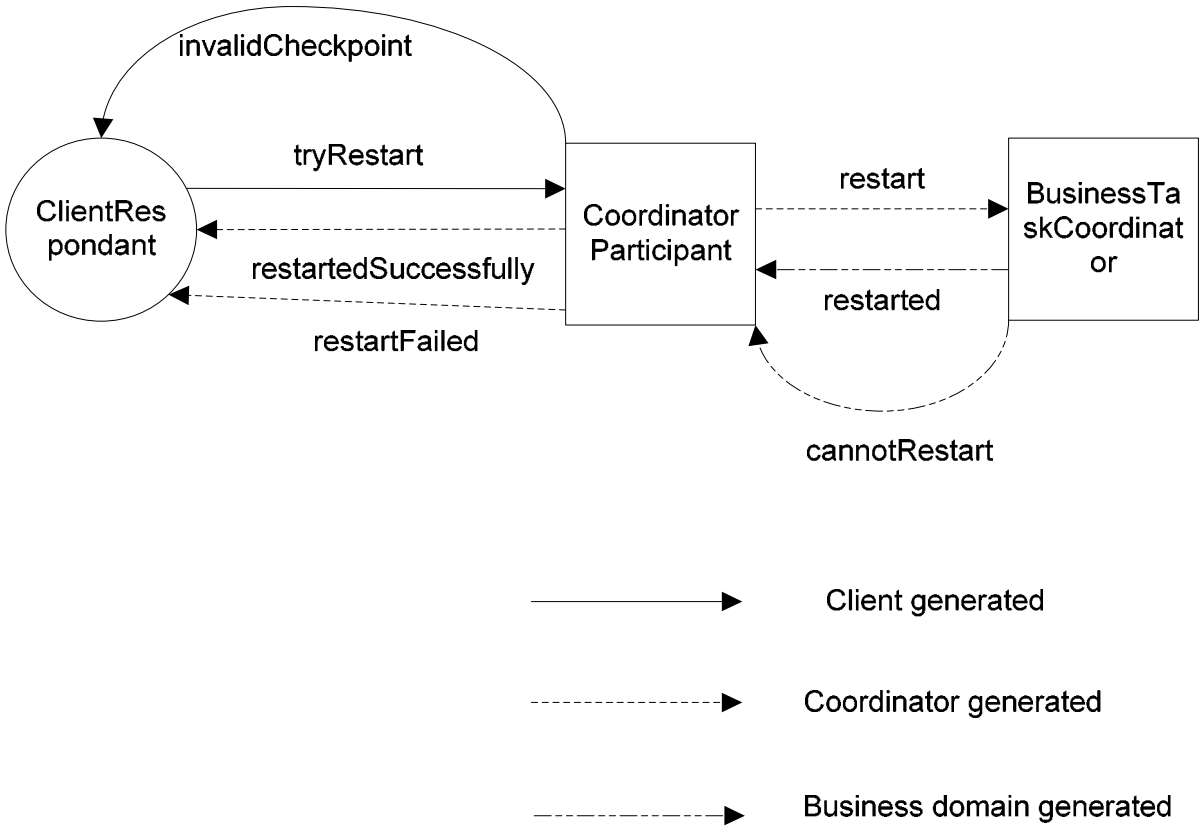

Figure 33, Restart protocol interactions.

The actor definitions for the Restart protocol are shown in Figure 33.

```
<wsdl:portType name="ClientRespondantPortType">
  <wsdl:operation name="invalidCheckpoint">
    <wsdl:input message="tns:InvalidCheckpointMessage"/>
  </wsdl:operation>
  <wsdl:operation name="restartedSuccessfully">
    <wsdl:input message="tns:RestartedSuccessfullyMessage"/>
  </wsdl:operation>
  <wsdl:operation name="restartFailed">
    <wsdl:input message="tns:RestartFailedMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="CoordinatorParticipantPortType">
  <wsdl:operation name="tryRestart">
    <wsdl:input message="tns:TryRestartMessage"/>
  </wsdl:operation>
```

```
  <wsdl:operation name="restarted">
    <wsdl:input message="tns:RestartedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="cannotRestart">
    <wsdl:input message="tns:CannotRestartMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="BusinessTaskCoordinatorPortType">
  <wsdl:operation name="restart">
    <wsdl:input message="tns:RestartMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

Figure 34, The Restart Protocol Actor WSDL portType Declarations.

An application uses the *workStatus* protocol to determine whether or not the business process as a whole is able to complete successfully, such that all work requested has been performed and can be confirmed (the business process has entered a completion state). For example, in the home entertainment example, the user may invoke this protocol to determine whether all of the individual components have been procured and are awaiting final confirmation of the order. The BP sub-protocol URI for the *workStatus* protocol is http://www.webservicestransactions.org/wstxm/tx-bp/ws/2003/03 and this is used in the *addParticipant* invocation. The *workStatus* protocol can be executed by the application as many times as necessary by the application within the business process by sending the appropriate WS-CF *coordinate* message to the coordinator.

The Participant accepts the following messages (illustrated in Figure 35); the CoordinatorParticipant end-point address as defined in WS-CF is propagated on all messages:

- *getWorkStatus*: this message is sent to each business domain to determine the current status of the work (task). If the task is ready to complete in a successful manner then the BusinessTaskCoordinator responds with the *workStatusCompleted* message. If the task has cancelled its work then the BusinessTaskCoordinator responds with the *workStatusCancelled* message and the completion status of the activity is set to FAIL_ONLY. If the task is still processing its work then it responds with the *workStatusProcessing* message. The task may provide additional Qualifiers on the *workStatusCompleted* or *workStatusProcessing* messages, for example to indicate how much longer the task may require in order to complete processing.

Because the work performed by each business domain may take arbitrary amounts of time to complete, it is permissible for each domain to autonomously inform the coordinator when it is complete (in either a success or failure condition). A completed business domain sends a message to the coordinator indicating either *workCompleted* or *workCancelled*.

If an importing business domain completes its work on the first request then it is possible to optimize the protocol and indicate to the coordinator that this has occurred. To do so, the business domain can specify one of the Qualifiers described in 4.4.4 during interposition.

If all of the enlisted BusinessTaskCoordinators inform the coordinator that they have completed, then the coordinator will use the *terminate-notification* protocol described earlier.

If all of the business domains indicate they are *ready* to complete then the *workCompleted* message is sent to the ClientRespondant. If at least one domain is not yet ready then the *workProcessing* message is sent. Alternatively, if all of the domains have cancelled their work, the *workCancelled* message is sent to the ClientRespondant and the activity completion status is set to FAIL_ONLY.

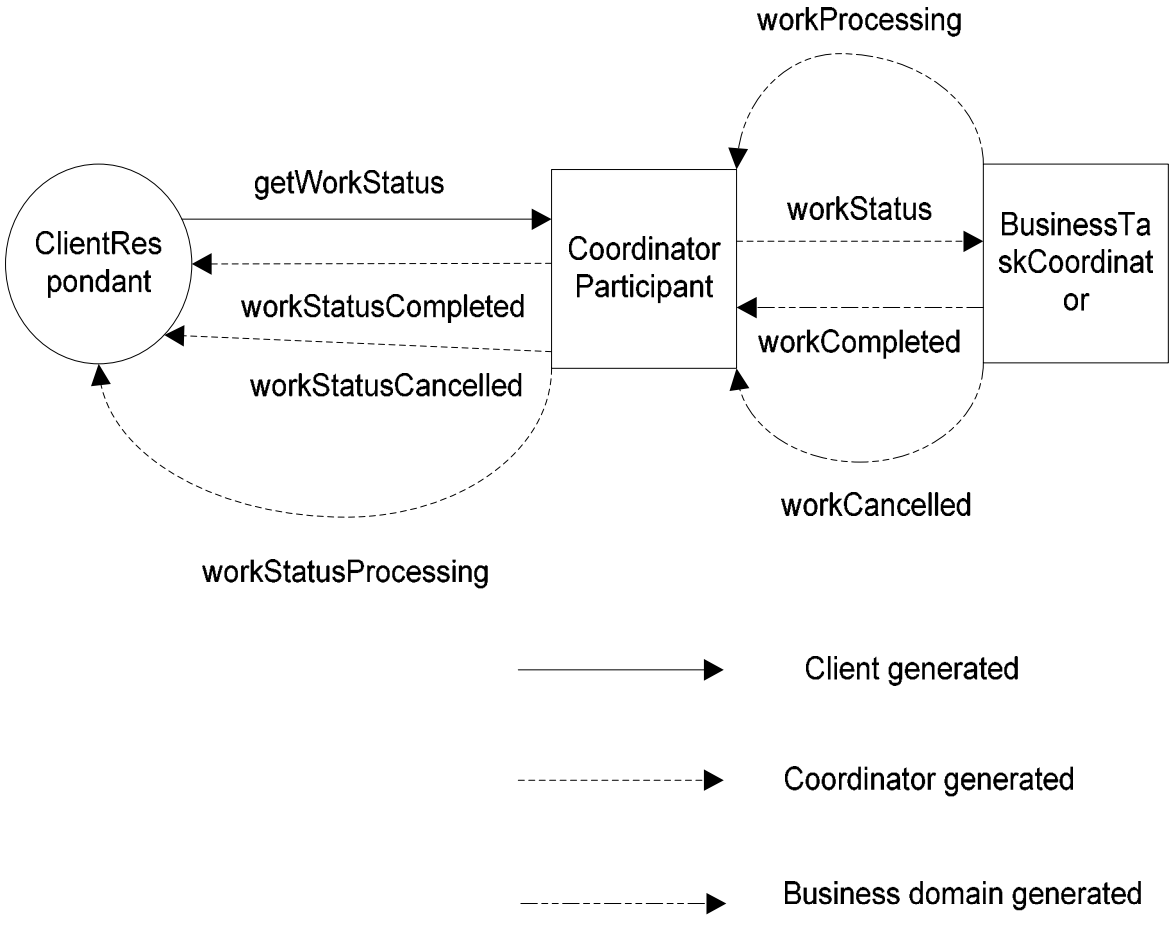

Figure 35, Status of work status protocol interactions.

The actor definitions for the Work Status protocol are shown in Figure 36.

```
<wsdl:portType name="ClientRespondantPortType">
  <wsdl:operation name="workStatusCompleted">
    <wsdl:input message="tns:WorkStatusCompletedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="workStatusCancelled">
    <wsdl:input message="tns:WorkStatusCancelledMessage"/>
  </wsdl:operation>
  <wsdl:operation name="workStatusProcessing">
    <wsdl:input message="tns:WorkStatusProcessingMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="CoordinatorParticipantPortType">
  <wsdl:operation name="getWorkStatus">
    <wsdl:input message="tns:GetWorkStatusMessage"/>
  </wsdl:operation>
  <wsdl:operation name="workProcessing">
    <wsdl:input message="tns:WorkProcessingMessage"/>
  </wsdl:operation>
  <wsdl:operation name="workCompleted">
    <wsdl:input message="tns:WorkCompletedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="workCancelled">
    <wsdl:input message="tns:WorkCancelledMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="BusinessTaskCoordinatorPortType">
  <wsdl:operation name="workStatus">
    <wsdl:input message="tns:WorkStatusMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

Figure 36, The Work Protocol Actor WSDL portType Declarations.

The BP sub-protocol URI for the *completion* protocol is http://www.webservicestransactions.org/wstxm/tx-bp/completion/2003/03 and this is

used in the *addParticipant* invocation. The *completion* protocol can only be executed once when the WS-CTX activity terminates. It is illegal to send the coordinate message to the coordinator specifying this protocol URI. If this is attempted, a compliant coordinator will return the *notCoordinated* message.

The Participant accepts the following messages (illustrated in Figure 37).

- *confirm*: the coordinator sends this message if the CompletionStatus of the business activity is SUCCESS. The business domain is required to confirm all work that it has performed in the scope of this business process. If successful, the *confirmed* message is sent to the CoordinatorParticipant. If the work cannot be confirmed then it should be cancelled (undone) and the *cancelled* message is sent. Alternatively, if the inability to complete the work is a transient, the *confirming* message is sent to the coordinator and the coordinator must enquire as to the eventual result; both the coordinator and the business domain must retain sufficient durable information to ensure this can occur despite failures. If it is not possible to determine whether or not the work has been completed (or in what state), the *unknownResult* message is sent.
- *cancel*: the coordinator sends this message if the CompletionStatus of the business activity is FAIL or FAIL_ONLY. The business domain is required to undo all work that it has performed in the scope of this business process. If successful, the *cancelled* message is sent to the CoordinatorParticipant. If the work cannot be cancelled and was in fact performed, then the confirmed message is sent back to the coordinator. Alternatively, if the inability to cancel the work is a transient, the *cancelling* message is returned by the business domain and the coordinator must enquire as to the eventual result; both the coordinator and the business domain must retain sufficient durable information to ensure this can occur despite failures. If it is not possible to determine whether or not the work has been completed (or in what state), the *unknownResult* message is sent.

If *all* of the business process work is successfully performed, a *processConfirmed* message is ultimately sent to the ClientRespondant. If *all* of the work is undone, then the *processCancelled* message is sent. The inability to determine the results of all business domains will cause the *unknownResultOccurred* message to be sent and the identifiers for all such domains will be included in the message. If some domains cancelled whilst other confirmed, the *mixedResponse* message will be sent and will contain the identifiers of each domain that cancelled and each domain that confirmed.

Error! Objects cannot be created from editing field codes.

Figure 37, Completion protocol interactions.

The actor definitions for the Completion protocol are shown in Figure 37.

```
<wsdl:portType name="ClientRespondantPortType">
  <wsdl:operation name="unknownResultOccurred">
    <wsdl:input message="tns:UnknownResultOccurredMessage"/>
```

```
  </wsdl:operation>
  <wsdl:operation name="processConfirmed">
    <wsdl:input message="tns:ProcessConfirmedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="processCancelled">
    <wsdl:input message="tns:ProcessCancelledMessage"/>
  </wsdl:operation>
  <wsdl:operation name="mixedResponse">
    <wsdl:input message="tns:MixedResponseMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="CoordinatorParticipantPortType">
  <wsdl:operation name="confirmProcess">
    <wsdl:input message="tns:ConfirmProcessMessage"/>
  </wsdl:operation>
  <wsdl:operation name="cancelProcess">
    <wsdl:input message="tns:CancelProcessMessage"/>
  </wsdl:operation>
  <wsdl:operation name="confirming">
    <wsdl:input message="tns:ConfirmingMessage"/>
  </wsdl:operation>
  <wsdl:operation name="confirmed">
    <wsdl:input message="tns:ConfirmedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="cancelled">
    <wsdl:input message="tns:CancelledMessage"/>
  </wsdl:operation>
  <wsdl:operation name="unknownResult">
    <wsdl:input message="tns:UnknownResultMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="BusinessTaskCoordinationPortType">
  <wsdl:operation name="confirm">
```

```
    <wsdl:input message="tns:ConfirmMessage"/>
  </wsdl:operation>
  <wsdl:operation name="cancel">
    <wsdl:input message="tns:CancelMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

Figure 38, The Completion Protocol Actor WSDL portType Declarations.

It is expected that the receipt of *cancelling*, *confirming* or *unknownResult* will be handled by the application or logged if not.

Note, a BusinessTaskCoordinator can resign from the business process at any time prior to the completion of an activity by sending the *removeParticipant* message to the coordinator.

4.4.3.3 Business process entities

Figure 39 illustrates how the various entities (end points) we have discussed in relation to the BP model may fit into the various business domains that constitute the overall business process. As mentioned before, this structuring allows for the federation of a business process. This means that although different implementations or coordination protocols may be used within a domain (federated space), the BusinessProcessParticipant translates to and from the BP protocol.

As shown, the TerminatorParticipant will typically reside at the root of the business process; this may be the ultimate root (e.g., the client) or may be a sub-root for nested business tasks.

BusinessProcessParticipants may be registered with child tasks in order to provide a level of business-fault tolerance. Each sub-domain (task) has a BusinessTaskCoordinator that ties it into the overall business process.

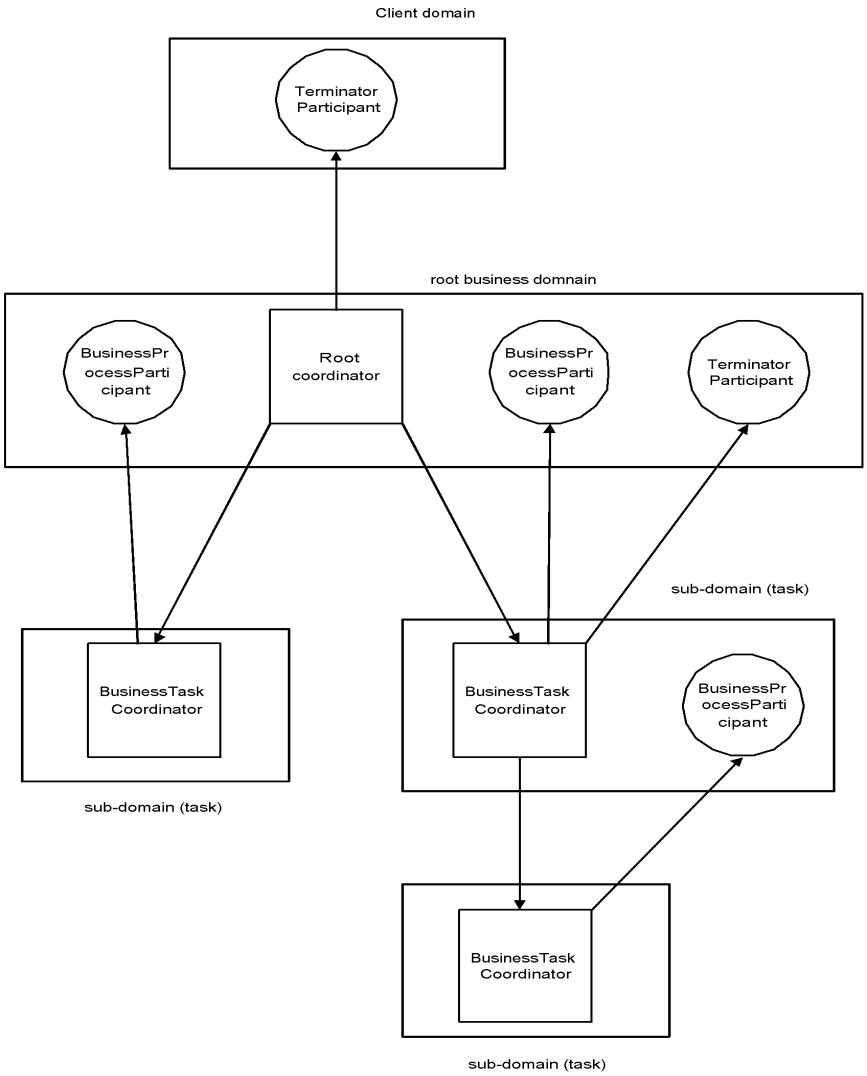

Figure 39, Example business process entities.

4.4.4 Qualifers

When a BusinessTaskCoordinator is enrolled with a business process, the entity performing the enrol can supply a number of qualifiers which may be used by the coordinator and business application to influence the overall outcome of the activity. The currently supported qualifiers are:

- *workCompleted*: the task is ready to complete in a successful manner. This is equivalent to the business domain sending the *workCompleted setResponse* message.
- *workCancelled*: the task has cancelled its work; the completion status of the activity is set to FAIL_ONLY. This is equivalent to the business domain sending the *workCancelled setResponse* message.

- *checkpointTimelimit*: it is likely that each checkpoint that is taken may be valid for only a specific period of time. This Qualifier is used to indicate (in seconds) how long a specific checkpoint can be relied upon. The time limit is not meant to be a guarantee on the lifecycle of a checkpoint, but while the time limit has not elapsed, the domain is required to retain the checkpoint with best effort.

4.4.5 Status values

The following extensions to the WS-CTX Status type are returned by business domains or their participants to indicate the current status:

- Working: the domain is performing the business task.
- Checkpointing: the domain is currently performing a checkpoint operation.
- Checkpointed: the domain has successfully checkpointed.
- Restarting: the domain is currently restarting from a saved checkpoint.
- Restarted: the domain has successfully restarted from a saved checkpoint.
- Cancelling: the domain is in the process of cancelling the business task.
- Cancelled: the domain has cancelled the business task.
- Confirming: the domain is in the process of confirming the business task.
- Confirmed: the domain has confirmed the business task.
- Failure: there has been a failure in the normal processing of the business task.

5. WSDL Interfaces and XML Schema Definitions

5.1 The WS-TXM Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema
targetNamespace="http://www.webservicestransactions.org/schemas/wstxm
/2003/03" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wscf="http://www.webservicestransactions.org/schemas/wscf/2003/
03"
xmlns:tns="http://www.webservicestransactions.org/schemas/wstxm/2003/
03">

  <xs:import
namespace="http://www.webservicestransactions.org/schemas/wscf/2003/0
3" schemaLocation="../../WS-CF/xml/wscf.xsd"/>

  <xs:complexType name="AssertionType">

    <xs:complexContent>

      <xs:extension base="wscf:AssertionType"/>

    </xs:complexContent>
```

```
  </xs:complexType>
  <xs:element name="assertion" type="tns:AssertionType"
abstract="true"/>
  <xs:complexType name="FaultType">
    <xs:complexContent>
      <xs:extension base="wscf:FaultType"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="fault" type="tns:FaultType" abstract="true"/>
  <xs:complexType name="ContextType">
    <xs:complexContent>
      <xs:extension base="wscf:ContextType">
        <xs:sequence>
          <xs:element name="timelimit" type="xs:positiveInteger"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="context" type="tns:ContextType"
substitutionGroup="wscf:context"/>
  <xs:complexType name="QualifierType">
    <xs:complexContent>
      <xs:extension base="wscf:QualifierType"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="qualifier" type="tns:QualifierType"
abstract="true" substitutionGroup="wscf:qualifier"/>
  <xs:simpleType name="StatusType">
    <xs:restriction base="wscf:StatusType"/>
  </xs:simpleType>
  <xs:element name="status" type="tns:StatusType"
substitutionGroup="wscf:status"/>
</xs:schema>
```

5.2 The tx-acid Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema
targetNamespace="http://www.webservicestransactions.org/schemas/wstxm
/tx-acid/2003/03"
xmlns:tns="http://www.webservicestransactions.org/schemas/wstxm/tx-
acid/2003/03" xmlns:
wstxm="http://www.webservicestransactions.org/schemas/wstxm/2003/03"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/2003/
03" schemaLocation="../wstxm.xsd"/>

  <xs:simpleType name="StatusType">

    <xs:restriction base="wstxm:StatusType">

      <xs:enumeration value="activity.status.tx-acid.ROLLBACK_ONLY"/>

      <xs:enumeration value="activity.status.tx-acid.ROLLING_BACK"/>

      <xs:enumeration value="activity.status.tx-acid.ROLLED_BACK"/>

      <xs:enumeration value="activity.status.tx-acid.COMMITTING"/>

      <xs:enumeration value="activity.status.tx-acid.COMMITTED"/>

      <xs:enumeration value="activity.status.tx-
acid.HEURISTIC_ROLLBACK"/>

      <xs:enumeration value="activity.status.tx-
acid.HEURISTIC_HAZARD"/>

      <xs:enumeration value="activity.status.tx-
acid.HEURISTIC_MIXED"/>

      <xs:enumeration value="activity.status.tx-acid.PREPARING"/>

      <xs:enumeration value="activity.status.tx-acid.PREPARED"/>

    </xs:restriction>

  </xs:simpleType>

  <xs:element name="status" type="tns:StatusType"
substitutionGroup="wstxm:status"/>

  <xs:complexType name="ContextType">

    <xs:complexContent>

      <xs:extension base="wstxm:ContextType"/>

    </xs:complexContent>

  </xs:complexType>
```

```
  <xs:element name="context" type="tns:ContextType"/>
  <xs:complexType name="AssertionType">
    <xs:complexContent>
      <xs:extension base="wstxm:AssertionType"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="FaultType">
    <xs:complexContent>
      <xs:extension base="wstxm:FaultType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

5.3 2PC Protocol

5.3.1 WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
targetNamespace="http://www.webservicestransactions.org/wsdl/wstxm/tx
-acid/2pc/2003/03"
xmlns:tns="http://www.webservicestransactions.org/wsdl/wstxm/tx-
acid/2pc/2003/03"
xmlns:2pc="http://www.webservicestransactions.org/schemas/wstxm/2pc/2
003/03" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/2pc/2
003/03" location="2pc.xsd"/>
  <!-- 2PC protocol messages -->
  <wsdl:message name="ContextMessage">
    <wsdl:part name="content" element="2pc:context"/>
  </wsdl:message>
  <wsdl:message name="PrepareMessage">
    <wsdl:part name="content" element="2pc:prepare"/>
  </wsdl:message>
  <wsdl:message name="VoteMessage">
    <wsdl:part name="content" element="2pc:vote"/>
```

```
  </wsdl:message>
  <wsdl:message name="RollbackMessage">
    <wsdl:part name="content" element="2pc:rollback"/>
  </wsdl:message>
  <wsdl:message name="RolledBackMessage">
    <wsdl:part name="content" element="2pc:rolled-back"/>
  </wsdl:message>
  <wsdl:message name="CommitMessage">
    <wsdl:part name="content" element="2pc:commit"/>
  </wsdl:message>
  <wsdl:message name="CommittedMessage">
    <wsdl:part name="content" element="2pc:committed"/>
  </wsdl:message>
  <wsdl:message name="OnePhaseCommitMessage">
    <wsdl:part name="content" element="2pc:one-phase-commit"/>
  </wsdl:message>
  <wsdl:message name="ForgetHeuristicMessage">
    <wsdl:part name="content" element="2pc:forget-heuristic"/>
  </wsdl:message>
  <wsdl:message name="HeuristicForgottenMessage">
    <wsdl:part name="content" element="2pc:heuristic-forgotten"/>
  </wsdl:message>
  <!-- 2PC protocol fault messages -->
  <wsdl:message name="HeuristicFaultMessage">
    <wsdl:part name="content" element="2pc:heuristic"/>
  </wsdl:message>
  <!-- 2PC protocol actor portType declarations -->
  <wsdl:portType name="twoPCParticipantPortType">
    <wsdl:operation name="prepare">
      <wsdl:input message="tns:PrepareMessage"/>
    </wsdl:operation>
    <wsdl:operation name="onePhaseCommit">
      <wsdl:input message="tns:OnePhaseCommitMessage"/>
```

```
    </wsdl:operation>
    <wsdl:operation name="rollback">
      <wsdl:input message="tns:RollbacklMessage"/>
    </wsdl:operation>
    <wsdl:operation name="commit">
      <wsdl:input message="tns:CommitMessage"/>
    </wsdl:operation>
    <wsdl:operation name="forgetHeuristic">
      <wsdl:input message="tns:ForgetHeuristicMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="CoordinatorParticipantPortType">
    <wsdl:operation name="committed">
      <wsdl:input message="tns:CommittedMessage"/>
    </wsdl:operation>
    <wsdl:operation name="rolledBack">
      <wsdl:input message="tns:RolledBackMessage"/>
    </wsdl:operation>
    <wsdl:operation name="vote">
      <wsdl:input message="tns:VoteMessage"/>
    </wsdl:operation>
    <wsdl:operation name="heuristicForgotten">
      <wsdl:input message="tns:HeuristicForgottenMessage"/>
    </wsdl:operation>
    <wsdl:operation name="heuristicFault">
      <wsdl:input message="tns:HeuristicFaultMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <!-- 2PC protocol actor SOAP bindings -->
  <wsdl:binding name="twoPCParticipantPortTypeSOAPBinding"
type="tns:twoPCParticipantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
```

```
    <wsdl:operation name="prepare">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
acid/2pc/2003/03/prepare" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="onePhaseCommit">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
acid/2pc/2003/03/onePhaseCommit" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="rollback">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
acid/2pc/2003/03/rollback" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="commit">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
acid/2pc/2003/03/commit" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
```

```
    </wsdl:operation>
    <wsdl:operation name="forgetHeuristic">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
acid/2pc/2003/03/forgetHeuristic" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="CoordinatorParticipantPortTypeSOAPBinding"
type="tns:CoordinatorParticipantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="committed">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
acid/2pc/2003/03/committed" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="rolledBack">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
acid/2pc/2003/03/rolledBack" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="vote">
```

```
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
acid/2pc/2003/03/vote" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="heuristicForgotten">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
acid/2pc/2003/03/heuristicForgotten" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="heuristicFault">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
acid/2pc/2003/03/heuristicFault" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

5.3.2 Schema

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xs:schema
targetNamespace="http://www.webservicestransactions.org/schemas/wstxm
/tx-acid/2pc/2003/03"
xmlns:tns="http://www.webservicestransactions.org/schemas/wstxm/tx-
acid/2pc/2003/03"
xmlns:acid="http://www.webservicestransactions.org/schemas/wstxm/tx-
acid/2003/03" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/tx-
acid/2003/03" schemaLocation="../tx-acid.xsd"/>
  <xs:element name="context">
    <xs:complexType>
      <xs:complexContent>
        <xs:restriction base="acid:ContextType"/>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="AssertionType">
    <xs:complexContent>
      <xs:extension base="acid:AssertionType">
        <xs:sequence>
          <xs:element name="participant-id" type="xs:anyURI"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="FaultType">
    <xs:complexContent>
      <xs:extension base="acid:FaultType">
        <xs:sequence>
          <xs:element name="participant-id" type="xs:anyURI"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

```
<xs:element name="prepare" type="tns:AssertionType"/>
<xs:element name="vote">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="tns:AssertionType">
        <xs:sequence>
          <xs:element name="decision">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="READ_ONLY"/>
                <xs:enumeration value="COMMIT"/>
                <xs:enumeration value="ROLLBACK"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:element name="heuristic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="tns:FaultType">
        <xs:sequence>
          <xs:element name="heuristic-type">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="HAZARD"/>
                <xs:enumeration value="MIXED"/>
                <xs:enumeration value="COMMIT"/>
                <xs:enumeration value="ROLLBACK"/>
              </xs:restriction>
```

```
              </xs:simpleType>
            </xs:element>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="rollback" type="tns:AssertionType"/>
  <xs:element name="rolled-back" type="tns:AssertionType"/>
  <xs:element name="commit" type="tns:AssertionType"/>
  <xs:element name="committed" type="tns:AssertionType"/>
  <xs:element name="one-phase-commit" type="tns:AssertionType"/>
  <xs:element name="forget-heuristic" type="tns:AssertionType"/>
  <xs:element name="heuristic-forgotten" type="tns:AssertionType"/>
</xs:schema>
```

5.4 Sync Protocol

5.4.1 WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
targetNamespace="http://www.webservicestransactions.org/wsdl/wstxm/tx
-acid/sync/2003/03"
xmlns:tns="http://www.webservicestransactions.org/wsdl/wstxm/tx-
acid/sync/2003/03"
xmlns:sync="http://www.webservicestransactions.org/schemas/wstxm/tx-
acid/sync/2003/03" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/tx-
acid/sync/2003/03" location="sync.xsd"/>
  <!-- Synchronisation protocol messages -->
  <wsdl:message name="ContextMessage">
    <wsdl:part name="content" element="sync:context"/>
  </wsdl:message>
  <wsdl:message name="BeforeCompletionMessage">
    <wsdl:part name="content" element="sync:before-completion"/>
```

```
  </wsdl:message>
  <wsdl:message name="BeforeCompletionParticipantRegisteredMessage">
    <wsdl:part name="content" element="sync:before-completion-
participant-registered"/>
  </wsdl:message>
  <wsdl:message name="AfterCompletionMessage">
    <wsdl:part name="content" element="sync:after-completion"/>
  </wsdl:message>
  <wsdl:message name="AfterCompletionParticipantRegisteredMessage">
    <wsdl:part name="content" element="sync:after-completion-
participant-registered"/>
  </wsdl:message>
  <!-- Sync protocol actor portType declarations -->
  <wsdl:portType name="SynchronizationPortType">
    <wsdl:operation name="beforeCompletion">
      <wsdl:input message="tns:BeforeCompletionMessage"/>
    </wsdl:operation>
    <wsdl:operation name="afterCompletion">
      <wsdl:input message="tns:AfterCompletionMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="CoordinatorParticipantPortType">
    <wsdl:operation name="beforeCompletionParticipantRegistered">
      <wsdl:input
message="tns:BeforeCompletionParticipantRegisteredMessage"/>
    </wsdl:operation>
    <wsdl:operation name="afterCompletionParticipantRegistered">
      <wsdl:input
message="tns:AfterCompletionParticipantRegisteredMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <!-- SOAP binding for sync protocol actors -->
  <wsdl:binding name="SynchronizationPortTypeSOAPBinding"
type="tns:SynchronizationPortType">
```

```
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="beforeCompletion">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
acid/sync/2003/03/beforeCompletion" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="afterCompletion">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
acid/sync/2003/03/afterCompletion" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="CoordinatorParticipantPortTypeSOAPBinding"
type="tns:CoordinatorParticipantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="beforeCompletionParticipantRegistered">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
acid/sync/2003/03/beforeCompletionParticipantRegistered"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="afterCompletionParticipantRegistered">
```

```
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
acid/sync/2003/03/afterCompletionParticipantRegistered"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

5.4.2 Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
targetNamespace="http://www.webservicestransactions.org/schemas/wstxm
/tx-acid/sync/2003/03"
xmlns:tns="http://www.webservicestransactions.org/schemas/wstxm/tx-
acid/sync/2003/03"
xmlns:acid="http://www.webservicestransactions.org/schemas/wstxm/tx-
acid/2003/03" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/tx-
acid/2003/03" schemaLocation="../tx-acid.xsd"/>
  <xs:complexType name="AssertionType">
    <xs:complexContent>
      <xs:extension base="acid:AssertionType">
        <xs:sequence>
          <xs:element ref="acid:status"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="context">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="acid:ContextType"/>
```

```
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="before-completion" type="tns:AssertionType"/>
  <xs:element name="before-completion-participant-registered"
type="acid:AssertionType"/>
  <xs:element name="after-completion" type="tns:AssertionType"/>
  <xs:element name="after-completion-participant-registered"
type="acid:AssertionType"/>
</xs:schema>
```

5.5 The TX-LRA Protocol

5.5.1 WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
targetNamespace="http://www.webservicestransactions.org/wsdl/wstxm/tx
-lra/2003/03"
xmlns:tns="http://www.webservicestransactions.org/wsdl/wstxm/tx-
lra/2003/03" xmlns:tx-
lra="http://www.webservicestransactions.org/schemas/wstxm/tx-
lra/2003/03" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/tx-
lra/2003/03" location="lra.xsd"/>
  <wsdl:message name="ContextMessage">
    <wsdl:part name="content" element="tx-lra:context"/>
  </wsdl:message>
  <!-- Outcome messages to participant -->
  <wsdl:message name="FailMessage"/>
  <wsdl:message name="FailedMessage"/>
  <wsdl:message name="SuccessMessage"/>
  <wsdl:message name="SucceededMessage"/>
  <!-- LRA protocol messages -->
  <wsdl:message name="CompensateMessage"/>
  <wsdl:message name="CompensatedMessage"/>
```

```
<wsdl:message name="CompleteMessage"/>
<wsdl:message name="CompletedMessage"/>
<wsdl:message name="ForgetMessage"/>
<wsdl:message name="ForgotMessage"/>
<!-- LRA protocol fault messages -->
<wsdl:message name="UnknownCompensatorFaultMessage"/>
<wsdl:message name="CannotCompensateFaultMessage"/>
<wsdl:message name="CannotCompleteFaultMessage"/>
<!-- LRA protocol actor portType declarations -->
<wsdl:portType name="CompensatorPortType">
  <wsdl:operation name="compensate">
    <wsdl:input message="tns:CompensateMessage"/>
  </wsdl:operation>
  <wsdl:operation name="complete">
    <wsdl:input message="tns:CompleteMessage"/>
  </wsdl:operation>
  <wsdl:operation name="forget">
    <wsdl:input message="tns:ForgetMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="CoordinatorPortType">
  <wsdl:operation name="compensated">
    <wsdl:input message="tns:CompensatedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="completed">
    <wsdl:input message="tns:CompletedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="forgot">
    <wsdl:input message="tns:ForgotMessage"/>
  </wsdl:operation>
  <wsdl:operation name="unknownCompensator">
    <wsdl:input message="tns:UnknownCompensatorFaultMessage"/>
  </wsdl:operation>
```

```
    <wsdl:operation name="cannotCompensate">
      <wsdl:input message="tns:CannotCompensateFaultMessage"/>
    </wsdl:operation>
    <wsdl:operation name="cannotComplete">
      <wsdl:input message="tns:CannotCompleteFaultMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <!-- Bindings for LRA protocol actors -->
  <wsdl:binding name="CompensatorPortTypeSOAPBinding"
type="tns:CompensatorPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="compensate">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
lra/2003/03/compensate" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="complete">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
lra/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="forget">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
lra/2003/03/forget" style="document"/>
      <wsdl:input>
```

```
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="CoordinatorPortTypeSOAPBinding"
type="tns:CoordinatorPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="compensated">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
lra/2003/03/compensated" 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/wstxm/tx-
lra/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="forgot">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
lra/2003/03/forgot" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
```

```
    </wsdl:operation>
    <wsdl:operation name="unknownCompensator">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
lra/2003/03/unknownCompensator" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="cannotCompensate">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
lra/2003/03/cannotCompensate" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="cannotComplete">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
lra/2003/03/cannotComplete" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

5.5.2 Schema

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xs:schema
targetNamespace="http://www.webservicestransactions.org/schemas/wstxm
/tx-lra/2003/03" xmlns:
wstxm="http://www.webservicestransactions.org/schemas/wstxm/2003/03"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://www.webservicestransactions.org/schemas/wstxm/tx-
lra/2003/03">

  <xs:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/2003/
03" schemaLocation="../wstxm.xsd"/>

  <xs:element name="context">

    <xs:complexType>

      <xs:complexContent>

        <xs:extension base="wstxm:ContextType">

          <xs:sequence>

            <xs:element name="lra-id" type="xs:anyURI"/>

            <xs:element name="coordinator-hierarchy">

              <xs:complexType>

                <xs:sequence>

                  <xs:element name="coordinator-location"
type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>

                </xs:sequence>

              </xs:complexType>

            </xs:element>

          </xs:sequence>

        </xs:extension>

      </xs:complexContent>

    </xs:complexType>

  </xs:element>

  <xs:element name="TimeLimitQualifier"
substitutionGroup="wstxm:qualifier">

    <xs:complexType>

      <xs:complexContent>

        <xs:extension base="wstxm:QualifierType">

          <xs:sequence>

            <xs:element name="compensator" type="xs:anyURI"/>
```

```
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="StatusType">
    <xs:restriction base="wstxm:StatusType">
      <xs:enumeration value="Compensating"/>
      <xs:enumeration value="Compensated"/>
      <xs:enumeration value="Completing"/>
      <xs:enumeration value="Completed"/>
      <xs:enumeration value="FailedToComplete"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="status" type="tns:StatusType"
substitutionGroup="wstxm:status"/>
  <xs:complexType name="AssertionType">
    <xs:complexContent>
      <xs:extension base="wstxm:AssertionType">
        <xs:sequence>
          <xs:element name="participant-id" type="xs:anyURI"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="forget" type="tns:AssertionType"/>
  <xs:element name="compensate" type="tns:AssertionType"/>
  <xs:element name="complete" type="tns:AssertionType"/>
  <xs:element name="forgot" type="tns:AssertionType"/>
  <xs:element name="compensated" type="tns:AssertionType"/>
  <xs:element name="completed" type="tns:AssertionType"/>
  <xs:element name="unknown-compensator-fault"
type="wstxm:FaultType"/>
```

```
  <xs:element name="cannot-compensate-fault" type="wstxm:FaultType"/>
  <xs:element name="cannot-complete-fault" type="wstxm:FaultType"/>
</xs:schema>
```

5.6 The TX-BP Protocol

5.6.1 Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
targetNamespace="http://www.webservicestransactions.org/schemas/wstxm
/tx-bp/2003/03" xmlns:
wstxm="http://www.webservicestransactions.org/schemas/wstxm/2003/03"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/2003/03">
  <xs:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/2003/
03" schemaLocation="../wstxm.xsd"/>
  <xs:complexType name="AssertionType">
    <xs:complexContent>
      <xs:extension base="wstxm:AssertionType">
        <xs:sequence>
          <xs:element name="participant-id" type="xs:anyURI"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="ContextType">
    <xs:complexContent>
      <xs:extension base="wstxm:ContextType">
        <xs:sequence>
          <xs:element name="process-id" type="xs:anyURI"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

```
  <xs:complexType name="QualifierType">
    <xs:complexContent>
      <xs:extension base="wstxm:QualifierType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

5.7 The BP Protocol

5.7.1 WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
targetNamespace="http://www.webservicestransactions.org/wsdl/wstxm/tx
-bp/bp/2003/03"
xmlns:tns="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/bp/2003/03"
xmlns:bp="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/bp/2003/03" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/bp/2003/03" location="bp.xsd"/>
  <!-- Business process protocol messages -->
  <wsdl:message name="FailureMessage">
    <wsdl:part name="content" element="bp:failure"/>
  </wsdl:message>
  <wsdl:message name="FailureAcknowledgedMessage">
    <wsdl:part name="content" element="bp:failure-acknowledged"/>
  </wsdl:message>
  <wsdl:message name="FailureHazardMessage">
    <wsdl:part name="content" element="bp:failure-hazard"/>
  </wsdl:message>
  <wsdl:message name="FailureHazardAcknowledgedMessage">
    <wsdl:part name="content" element="bp:failure-hazard-
acknowledged"/>
  </wsdl:message>
  <!-- BP protocol actors -->
```

```
  <wsdl:portType name="BusinessProcessParticipantPortType">
    <wsdl:operation name="failure">
      <wsdl:input message="FailureMessage"/>
    </wsdl:operation>
    <wsdl:operation name="failureHazard">
      <wsdl:input message="FailureHazardMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="CoordinatorParticipantPortType">
    <wsdl:operation name="failureAcknowledged">
      <wsdl:input message="FailureAcknowledgedMessage"/>
    </wsdl:operation>
    <wsdl:operation name="failureHazardAcknowledged">
      <wsdl:input message="FailureHazardAcknowledgedMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <!-- BP protocol actor SOAP bindings -->
  <wsdl:binding name="BusinessProcessParticipantPortTypeSOAPBinding"
type="tns:BusinessProcessParticipantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="failure">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/bp/2003/03/failure" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="wsc4c:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="failureHazard">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/bp/2003/03/failureHazard" style="document"/>
      <wsdl:input>
```

```
        <soap:body use="literal"/>
        <soap:header use="literal" message="wsc4c:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="CoordinatorParticipantPortTypeSOAPBinding"
type="tns:CoordinatorParticipantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="failureAcknowledged">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/bp/2003/03/failureAcknowledged" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="wsc4c:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="failureHazardAcknowledged">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/bp/2003/03/failureHazardAcknowledged" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="wsc4c:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

5.7.2 Schema

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xs:schema
targetNamespace="http://www.webservicestransactions.org/schemas/wstxm
/tx-bp/bp/2003/03"
xmlns:tns="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/bp/2003/03"
xmlns:bp="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/2003/03" xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/2003/03" schemaLocation="../bp.xsd"/>

  <xs:element name="failure" type="bp:AssertionType"/>

  <xs:element name="failure-acknowledged" type="bp:AssertionType"/>

  <xs:element name="failure-hazard" type="bp:AssertionType"/>

  <xs:element name="failure-hazard-acknowledged"
type="bp:AssertionType"/>

</xs:schema>
```

5.8 The Completion Protocol

5.8.1 WSDL

```
<?xml version="1.0" encoding="utf-8"?>

<wsdl:definitions
targetNamespace="http://www.webservicestransactions.org/wsdl/wstxm/tx
-bp/completion/2003/03"
xmlns:tns="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/completion/2003/03"
xmlns:comp="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/ws/2003/03" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

  <wsdl:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/ws/2003/03" location="completion.xsd"/>

  <!-- Completion protocol messages -->

  <wsdl:message name="ContextMessage">

    <wsdl:part name="content" element="comp:context"/>

  </wsdl:message>

  <wsdl:message name="UnknownResultOccurredMessage">

    <wsdl:part name="content" element="comp:unknown-result-
occurred"/>

  </wsdl:message>
```

```
<wsdl:message name="ConfirmProcessMessage">
  <wsdl:part name="content" element="comp:confirm-process"/>
</wsdl:message>
<wsdl:message name="ProcessConfirmedMessage">
  <wsdl:part name="content" element="comp:process-confirmed"/>
</wsdl:message>
<wsdl:message name="CancelProcessMessage">
  <wsdl:part name="content" element="comp:cancel-process"/>
</wsdl:message>
<wsdl:message name="ProcessCancelledMessage">
  <wsdl:part name="content" element="comp:process-cancelled"/>
</wsdl:message>
<wsdl:message name="MixedResponseMessage">
  <wsdl:part name="content" element="comp:mixed-response"/>
</wsdl:message>
<wsdl:message name="ConfirmingMessage">
  <wsdl:part name="content" element="comp:confirming"/>
</wsdl:message>
<wsdl:message name="ConfirmMessage">
  <wsdl:part name="content" element="comp:confirm"/>
</wsdl:message>
<wsdl:message name="CancelMessage">
  <wsdl:part name="content" element="comp:cancel"/>
</wsdl:message>
<wsdl:message name="ConfirmedMessage">
  <wsdl:part name="content" element="comp:confirmed"/>
</wsdl:message>
<wsdl:message name="CancelledMessage">
  <wsdl:part name="content" element="comp:cancelled"/>
</wsdl:message>
<wsdl:message name="UnknownResultMessage">
  <wsdl:part name="content" element="comp:unknown-result"/>
</wsdl:message>
```

```
<!-- Completion protocol actors -->
<wsdl:portType name="ClientRespondantPortType">
  <wsdl:operation name="unknownResultOccurred">
    <wsdl:input message="tns:UnknownResultOccurredMessage"/>
  </wsdl:operation>
  <wsdl:operation name="processConfirmed">
    <wsdl:input message="tns:ProcessConfirmedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="processCancelled">
    <wsdl:input message="tns:ProcessCancelledMessage"/>
  </wsdl:operation>
  <wsdl:operation name="mixedResponse">
    <wsdl:input message="tns:MixedResponseMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="CoordinatorParticipantPortType">
  <wsdl:operation name="confirmProcess">
    <wsdl:input message="tns:ConfirmProcessMessage"/>
  </wsdl:operation>
  <wsdl:operation name="cancelProcess">
    <wsdl:input message="tns:CancelProcessMessage"/>
  </wsdl:operation>
  <wsdl:operation name="confirming">
    <wsdl:input message="tns:ConfirmingMessage"/>
  </wsdl:operation>
  <wsdl:operation name="confirmed">
    <wsdl:input message="tns:ConfirmedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="cancelled">
    <wsdl:input message="tns:CancelledMessage"/>
  </wsdl:operation>
  <wsdl:operation name="unknownResult">
    <wsdl:input message="tns:UnknownResultMessage"/>
```

```
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="BusinessTaskCoordinationPortType">
    <wsdl:operation name="confirm">
      <wsdl:input message="tns:ConfirmMessage"/>
    </wsdl:operation>
    <wsdl:operation name="cancel">
      <wsdl:input message="tns:CancelMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <!-- Completion protocol actor SOAP bindings -->
  <wsdl:binding name="ClientRespondantPortTypeSOAPBinding"
type="tns:ClientRespondantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="unknownResultOccurred">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/completion/2003/03/unknownResultOccurred" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="processConfirmed">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/completion/2003/03/processConfirmed" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="processCancelled">
```

```
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/completion/2003/03/processCancelled" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="mixedResponse">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/completion/2003/03/mixedResponse" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="CoordinatorParticipantPortTypeSOAPBinding"
type="tns:CoordinatorParticipantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="confirmProcess">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/completion/2003/03/confirmProcess" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="cancelProcess">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/completion/2003/03/cancelProcess" style="document"/>
      <wsdl:input>
```

```
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="confirming">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/completion/2003/03/confirming" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="confirmed">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/completion/2003/03/confirmed" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="cancelled">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/completion/2003/03/cancelled" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="unknownResult">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/completion/2003/03/unknownResult" style="document"/>
```

```
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding
name="BusinessTaskCoordinationPortTypeTypeSOAPBinding"
type="tns:BusinessTaskCoordinationPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="confirm">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/completion/2003/03/confirm" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="cancel">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/completion/2003/03/cancel" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

5.8.2 Schema

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xs:schema
targetNamespace="http://www.webservicestransactions.org/schemas/wstxm
/tx-bp/completion/2003/03"
xmlns:tns="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/completion/2003/03"
xmlns:bp="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/2003/03" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/2003/03" schemaLocation="../bp.xsd"/>
  <xs:element name="context">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="bp:ContextType"/>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="AssertionType">
    <xs:complexContent>
      <xs:extension base="bp:AssertionType">
        <xs:sequence>
          <xs:element name="checkpoint-id" type="xs:anyURI"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="unknown-result-occurred">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element name="unknown" type="xs:anyURI"
maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:extension>
```

```
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="confirm-process" type="tns:AssertionType"/>
  <xs:element name="cancel-process" type="tns:AssertionType"/>
  <xs:element name="process-confirmed" type="tns:AssertionType"/>
  <xs:element name="process-cancelled" type="tns:AssertionType"/>
  <xs:element name="mixed-response">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:AssertionType">
          <xs:sequence>
            <xs:element name="confirmed" type="xs:anyURI"
maxOccurs="unbounded"/>
            <xs:element name="cancelled" type="xs:anyURI"
maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="confirming" type="tns:AssertionType"/>
  <xs:element name="confirm" type="tns:AssertionType"/>
  <xs:element name="cancel" type="tns:AssertionType"/>
  <xs:element name="confirmed" type="tns:AssertionType"/>
  <xs:element name="cancelled" type="tns:AssertionType"/>
</xs:schema>
```

5.9 The Checkpoint Protocol

5.9.1 WSDL

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<wsdl:definitions
targetNamespace="http://www.webservicestransactions.org/wsdl/wstxm/tx
-bp/cp/2003/03"
xmlns:tns="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/cp/2003/03"
xmlns:cp="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/cp/2003/03" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

  <wsdl:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/cp/2003/03" location="cp.xsd"/>

  <!-- CP Context -->

  <wsdl:message name="ContextMessage">

    <wsdl:part name="content" element="cp:context"/>

  </wsdl:message>

  <!-- CP protocol messages -->

  <wsdl:message name="CheckpointMessage">

    <wsdl:part name="content" element="cp:checkpoint"/>

  </wsdl:message>

  <wsdl:message name="CheckpointedMessage">

    <wsdl:part name="content" element="cp:checkpointed"/>

  </wsdl:message>

  <wsdl:message name="CheckpointFailedMessage">

    <wsdl:part name="content" element="cp:checkpoint-failed"/>

  </wsdl:message>

  <wsdl:message name="CheckpointingSucceededfulMessage">

    <wsdl:part name="content" element="cp:checkpointing-succeeded"/>

  </wsdl:message>

  <wsdl:message name="CheckpointingFailedMessage">

    <wsdl:part name="content" element="cp:checkpointing-failed"/>

  </wsdl:message>

  <wsdl:message name="CreateCheckpointMessage">

    <wsdl:part name="content" element="cp:create-checkpoint"/>

  </wsdl:message>

  <!-- CP protocol actors -->
```

```
  <wsdl:portType name="ClientRespondantPortType">
    <wsdl:operation name="checkpointingSuccedded">
      <wsdl:input message="tns:CheckpointingSucceededfulMessage"/>
    </wsdl:operation>
    <wsdl:operation name="checkpointingFailed">
      <wsdl:input message="tns:CheckpointingFailedMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="CoordinatorParticipantPortType">
    <wsdl:operation name="createCheckpoint">
      <wsdl:input message="tns:CreateCheckpointMessage"/>
    </wsdl:operation>
    <wsdl:operation name="checkpointed">
      <wsdl:input message="tns:CheckpointedMessage"/>
    </wsdl:operation>
    <wsdl:operation name="checkpointFailed">
      <wsdl:input message="tns:CheckpointFailedMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="BusinessTaskCoordinatorPortType">
    <wsdl:operation name="checkpoint">
      <wsdl:input message="tns:CheckpointMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <!-- CP protocol actor SOAP bindings -->
  <wsdl:binding name="ClientRespondantPortTypeSOAPBinding"
type="tns:ClientRespondantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="checkpointingSuccedded">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/cp/2003/03/checkpointingSuccedded" style="document"/>
      <wsdl:input>
```

```
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="checkpointingFailed">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/cp/2003/03/checkpointingFailed" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="CoordinatorParticipantPortTypeSOAPBinding"
type="tns:CoordinatorParticipantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="createCheckpoint">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/cp/2003/03/createCheckpoint" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="checkpointed">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/cp/2003/03/checkpointed" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
```

```
    </wsdl:operation>
    <wsdl:operation name="checkpointFailed">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/cp/2003/03/checkpointFailed" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="BusinessTaskCoordinatorPortTypeSOAPBinding"
type="tns:BusinessTaskCoordinatorPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="checkpoint">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/cp/2003/03/checkpoint" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

5.9.2 Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
targetNamespace="http://www.webservicestransactions.org/schemas/wstxm
/tx-bp/cp/2003/03"
xmlns:bp="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/2003/03"
xmlns:tns="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/cp/2003/03" xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
  <xs:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/2003/03" schemaLocation="../bp.xsd"/>
  <xs:element name="context">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="bp:ContextType"/>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="AssertionType">
    <xs:complexContent>
      <xs:extension base="bp:AssertionType">
        <xs:sequence>
          <xs:element name="checkpoint-id" type="xs:anyURI"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="QualifierType">
    <xs:complexContent>
      <xs:extension base="bp:QualifierType"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="create-checkpoint" type="tns:AssertionType"/>
  <xs:element name="checkpointing-failed" type="tns:AssertionType"/>
  <xs:element name="checkpointing-succeeded"
type="tns:AssertionType"/>
  <xs:element name="checkpoint" type="tns:AssertionType"/>
  <xs:element name="checkpointed" type="tns:AssertionType"/>
  <xs:element name="checkpoint-failed" type="tns:AssertionType"/>
  <xs:element name="checkpoint-timelimit-qualifier">
    <xs:complexType>
      <xs:complexContent>
```

```
        <xs:extension base="tns:QualifierType">
          <xs:sequence>
            <xs:element name="checkpoint-alive" type="xs:int"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

5.10 The Restart Protocol

5.10.1 WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
targetNamespace="http://www.webservicestransactions.org/wsdl/wstxm/tx
-bp/ws/2003/03"
xmlns:tns="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/ws/2003/03"
xmlns:rs="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/restart/2003/03" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/restart/2003/03" location="restart.xsd"/>
  <!-- Restart protocol messages -->
  <wsdl:message name="ContextMessage">
    <wsdl:part name="content" element="rs:context"/>
  </wsdl:message>
  <wsdl:message name="InvalidCheckpointMessage">
    <wsdl:part name="content" element="rs:invalid-checkpoint"/>
  </wsdl:message>
  <wsdl:message name="TryRestartMessage">
    <wsdl:part name="content" element="restart:try-restart"/>
  </wsdl:message>
  <wsdl:message name="RestartedSuccessfullyMessage">
    <wsdl:part name="content" element="rs:restarted-successfully"/>
```

```
  </wsdl:message>
  <wsdl:message name="RestartFailedMessage">
    <wsdl:part name="content" element="rs:restart-failed"/>
  </wsdl:message>
  <wsdl:message name="RestartMessage">
    <wsdl:part name="content" element="rs:restart"/>
  </wsdl:message>
  <wsdl:message name="RestartedMessage">
    <wsdl:part name="content" element="rs:restarted"/>
  </wsdl:message>
  <wsdl:message name="CannotRestartMessage">
    <wsdl:part name="content" element="rs:cannot-restarted"/>
  </wsdl:message>
  <!-- Restart protocol actors -->
  <wsdl:portType name="ClientRespondantPortType">
    <wsdl:operation name="invalidCheckpoint">
      <wsdl:input message="tns:InvalidCheckpointMessage"/>
    </wsdl:operation>
    <wsdl:operation name="restartedSuccessfully">
      <wsdl:input message="tns:RestartedSuccessfullyMessage"/>
    </wsdl:operation>
    <wsdl:operation name="restartFailed">
      <wsdl:input message="tns:RestartFailedMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="CoordinatorParticipantPortType">
    <wsdl:operation name="tryRestart">
      <wsdl:input message="tns:TryRestartMessage"/>
    </wsdl:operation>
    <wsdl:operation name="restarted">
      <wsdl:input message="tns:RestartedMessage"/>
    </wsdl:operation>
    <wsdl:operation name="cannotRestart">
```

```
      <wsdl:input message="tns:CannotRestartMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="BusinessTaskCoordinatorPortType">
    <wsdl:operation name="restart">
      <wsdl:input message="tns:RestartMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <!-- Restart protocol actor SOAP bindings -->
  <wsdl:binding name="ClientRespondantPortTypeSOAPBinding"
type="tns:ClientRespondantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="invalidCheckpoint">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/ws/2003/03/invalidCheckpoint" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="restartedSuccessfully">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/ws/2003/03/restartedSuccessfully" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="restartFailed">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/ws/2003/03/restartFailed" style="document"/>
```

```
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="CoordinatorParticipantPortTypeSOAPBinding"
type="tns:CoordinatorParticipantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="tryRestart">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/ws/2003/03/tryRestart" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="restarted">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/ws/2003/03/restarted" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="cannotRestart">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/ws/2003/03/cannotRestart" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
```

```
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="BusinessTaskCoordinatorPortTypeSOAPBinding"
type="tns:BusinessTaskCoordinatorPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="restart">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/ws/2003/03/restart" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

5.10.2 Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
targetNamespace="http://www.webservicestransactions.org/schemas/wstxm
/tx-bp/restart/2003/03" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:bp="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/2003/03"
xmlns:tns="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/restart/2003/03">
  <xs:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/2003/03" schemaLocation="../bp.xsd"/>
  <xs:element name="context">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="bp:ContextType"/>
      </xs:complexContent>
    </xs:complexType>
```

```
  </xs:element>
  <xs:complexType name="AssertionType">
    <xs:complexContent>
      <xs:extension base="bp:AssertionType">
        <xs:sequence>
          <xs:element name="checkpoint-id" type="xs:anyURI"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="invalid-checkpoint" type="tns:AssertionType"/>
  <xs:element name="try-restart" type="tns:AssertionType"/>
  <xs:element name="restarted-successfully"
type="tns:AssertionType"/>
  <xs:element name="restart-failed" type="tns:AssertionType"/>
  <xs:element name="restart" type="tns:AssertionType"/>
  <xs:element name="restarted" type="tns:AssertionType"/>
  <xs:element name="cannot-restart" type="tns:AssertionType"/>
</xs:schema>
```

5.11 The Terminate Notification Protocol

5.11.1 WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
targetNamespace="http://www.webservicestransactions.org/wsdl/wstxm/tx
-bp/tn/2003/03"
xmlns:tns="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/tn/2003/03"
xmlns:tn="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/tn/2003/03" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/tn/2003/03" location="tn.xsd"/>
  <!-- Terminate-notification protocol messages -->
  <wsdl:message name="ContextMessage">
```

```
    <wsdl:part name="content" element="tn:context"/>
  </wsdl:message>
  <wsdl:message name="ConfirmCompleteMessage">
    <wsdl:part name="content" element="tn:confirm-complete"/>
  </wsdl:message>
  <wsdl:message name="ConfirmCompletedMessage">
    <wsdl:part name="content" element="tn:confirm-completed"/>
  </wsdl:message>
  <wsdl:message name="CancelCompleteMessage">
    <wsdl:part name="content" element="tn:cancel-complete"/>
  </wsdl:message>
  <wsdl:message name="CancelCompletedMessage">
    <wsdl:part name="content" element="tn:cancel-completed"/>
  </wsdl:message>
  <!-- Terminate notification actor portType declarations -->
  <wsdl:portType name="TerminatorParticipantPortType">
    <wsdl:operation name="confirmComplete">
      <wsdl:input message="ConfirmCompleteMessage"/>
    </wsdl:operation>
    <wsdl:operation name="cancelComplete">
      <wsdl:input message="CancelCompleteMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="CoordinatorParticipantPortType">
    <wsdl:operation name="confirmCompleted">
      <wsdl:input message="ConfirmCompletedMessage"/>
    </wsdl:operation>
    <wsdl:operation name="cancelCompleted">
      <wsdl:input message="CancelCompletedMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <!-- Terminate notification actors SOAP binding -->
```

```
  <wsdl:binding name="TerminatorParticipantPortTypeSOAPBinding"
type="tns:TerminatorParticipantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="confirmComplete">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/tn/2003/03/confirmComplete" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="cancelComplete">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/tn/2003/03/cancelComplete" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="CoordinatorParticipantPortTypeSOAPBinding"
type="tns:CoordinatorParticipantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="confirmCompleted">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/tn/2003/03/confirmCompleted" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
```

```
    <wsdl:operation name="cancelCompleted">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/tn/2003/03/cancelCompleted" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

5.11.2 Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
targetNamespace="http://www.webservicestransactions.org/schemas/wstxm
/tx-bp/tn/2003/03"
xmlns:tns="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/tn/2003/03" xmlns:tx-
bp="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/2003/03" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/2003/03" schemaLocation="../bp.xsd"/>
  <xs:element name="context" type="tx-bp:ContextType"/>
  <xs:element name="confirm-complete" type="tx-bp:AssertionType"/>
  <xs:element name="confirm-completed" type="tx-bp:AssertionType"/>
  <xs:element name="cancel-complete" type="tx-bp:AssertionType"/>
  <xs:element name="cancel-completed" type="tx-bp:AssertionType"/>
</xs:schema>
```

5.12 The Work Status Protocol

5.12.1 WSDL

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<wsdl:definitions
targetNamespace="http://www.webservicestransactions.org/wsdl/wstxm/tx
-bp/ws/2003/03"
xmlns:tns="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/ws/2003/03"
xmlns:ws="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/ws/2003/03" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

  <wsdl:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/ws/2003/03" location="ws.xsd"/>

  <!-- Work status protocol messages -->

  <wsdl:message name="ContextMessage">

    <wsdl:part name="content" element="ws:context"/>

  </wsdl:message>

  <wsdl:message name="GetWorkStatusMessage">

    <wsdl:part name="content" element="ws:get-work-status"/>

  </wsdl:message>

  <wsdl:message name="WorkStatusCompletedMessage">

    <wsdl:part name="content" element="ws:work-status-completed"/>

  </wsdl:message>

  <wsdl:message name="WorkStatusCancelledMessage">

    <wsdl:part name="content" element="ws:work-status-cancelled"/>

  </wsdl:message>

  <wsdl:message name="WorkStatusProcessingMessage">

    <wsdl:part name="content" element="ws:work-status-processing"/>

  </wsdl:message>

  <wsdl:message name="WorkStatusMessage">

    <wsdl:part name="content" element="ws:work-status"/>

  </wsdl:message>

  <wsdl:message name="WorkCompletedMessage">

    <wsdl:part name="content" element="ws:work-completed"/>

  </wsdl:message>

  <wsdl:message name="WorkCancelledMessage">

    <wsdl:part name="content" element="ws:work-cancelled"/>
```

```
  </wsdl:message>
  <wsdl:message name="WorkProcessingMessage">
    <wsdl:part name="content" element="ws:work-processing"/>
  </wsdl:message>
  <!-- Work status protocol actors -->
  <wsdl:portType name="ClientRespondantPortType">
    <wsdl:operation name="workStatusCompleted">
      <wsdl:input message="tns:WorkStatusCompletedMessage"/>
    </wsdl:operation>
    <wsdl:operation name="workStatusCancelled">
      <wsdl:input message="tns:WorkStatusCancelledMessage"/>
    </wsdl:operation>
    <wsdl:operation name="workStatusProcessing">
      <wsdl:input message="tns:WorkStatusProcessingMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="CoordinatorParticipantPortType">
    <wsdl:operation name="getWorkStatus">
      <wsdl:input message="tns:GetWorkStatusMessage"/>
    </wsdl:operation>
    <wsdl:operation name="workProcessing">
      <wsdl:input message="tns:WorkProcessingMessage"/>
    </wsdl:operation>
    <wsdl:operation name="workCompleted">
      <wsdl:input message="tns:WorkCompletedMessage"/>
    </wsdl:operation>
    <wsdl:operation name="workCancelled">
      <wsdl:input message="tns:WorkCancelledMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="BusinessTaskCoordinatorPortType">
    <wsdl:operation name="workStatus">
      <wsdl:input message="tns:WorkStatusMessage"/>
```

```
    </wsdl:operation>
  </wsdl:portType>
  <!-- Work status protocol actor SOAP bindings -->
  <wsdl:binding name="ClientRespondantPortTypeSOAPBinding"
type="tns:ClientRespondantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="workStatusCompleted">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/ws/2003/03/workStatusCompleted" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="workStatusCancelled">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/ws/2003/03/workStatusCancelled" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="workStatusProcessing">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/ws/2003/03/workStatusProcessing" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
```

```
  <wsdl:binding name="CoordinatorParticipantPortTypeSOAPBinding"
type="tns:CoordinatorParticipantPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="getWorkStatus">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/ws/2003/03/getWorkStatus" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="workProcessing">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/ws/2003/03/workProcessing" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="workCompleted">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/ws/2003/03/workCompleted" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="workCancelled">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/ws/2003/03/workCancelled" style="document"/>
      <wsdl:input>
```

```
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="BusinessTaskCoordinatorPortTypeSOAPBinding"
type="tns:BusinessTaskCoordinatorPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="workStatus">
      <soap:operation
soapAction="http://www.webservicestransactions.org/wsdl/wstxm/tx-
bp/ws/2003/03/workStatus" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
        <soap:header use="literal" message="tns:ContextMessage"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

5.12.2 Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
targetNamespace="http://www.webservicestransactions.org/schemas/wstxm
/tx-bp/ws/2003/03"
xmlns:tns="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/ws/2003/03"
xmlns:bp="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/2003/03" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import
namespace="http://www.webservicestransactions.org/schemas/wstxm/tx-
bp/2003/03" schemaLocation="../bp.xsd"/>
  <xs:element name="context">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="bp:ContextType"/>
```

```
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="AssertionType">
    <xs:complexContent>
      <xs:extension base="bp:AssertionType">
        <xs:sequence>
          <xs:element ref="tns:work-completed-qualifier"
minOccurs="0"/>
          <xs:element ref="tns:work-cancelled-qualifier"
minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="QualifierType">
    <xs:complexContent>
      <xs:extension base="bp:QualifierType"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="get-work-status" type="tns:AssertionType"/>
  <xs:element name="work-status-completed" type="tns:AssertionType"/>
  <xs:element name="work-status-cancelled" type="tns:AssertionType"/>
  <xs:element name="work-status-processing"
type="tns:AssertionType"/>
  <xs:element name="work-processing" type="tns:AssertionType"/>
  <xs:element name="work-status" type="tns:AssertionType"/>
  <xs:element name="work-completed" type="tns:AssertionType"/>
  <xs:element name="work-cancelled" type="tns:AssertionType"/>
  <xs:element name="work-completed-qualifier">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:QualifierType"/>
      </xs:complexContent>
```

```
    </xs:complexType>
  </xs:element>
  <xs:element name="work-cancelled-qualifier">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:QualifierType"/>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

6. References

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

[2] J. N. Gray, “The transaction concept: virtues and limitations”, Proceedings of the 7th VLDB Conference, September 1981, pp. 144-154.

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.