Web Services Composite Application Framework (WS-CAF) 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)

INFORMATION THEREIN; THESE CHANGES WILL BE INCORPORATED INTO NEW VERSIONS OF THE SPECIFICATION, IF ANY. THE OWNERS MAY MAKE IMPROVEMENTS AND/OR CHANGES TO THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THE SPECIFICATION AT ANY TIME. Any use of such changes in the Specification will be governed by the then-current license for the applicable version of the Specification.

LIMITATION OF LIABILITY

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

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

Restricted Rights Legend

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

Report

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

Abstract
The ability to compose an application out of multiple Web services is a fundamental characteristic of the technology. However, without sufficient infrastructure support services, composite applications must rely on proprietary solutions to ensure correct operation.
Applications typically use multiple Web services in combination to solve specific business problems such as processing an order, updating multiple customer records from a single change request, or scheduling just in time arrival of parts for a manufacturing run.
The Web Services Composite Application Framework specifications, or WS-CAF, propose standard, interoperable mechanisms for managing shared context and ensuring business processes achieve predictable results and recovery from failure.
The ability to share system information, called context, is essential for the proper operation of applications composed of multiple Web services. Context identifies a Web service as part of a composite application and allows each Web service to share information such as message correlation, security tokens, and database connections.
Extensions to basic context operations provide the ability to support long running units of work such as business process automations and workflows. The WS-CAF specifications complement other Web services specifications in the areas of security, reliable messaging, choreography, and transactions. The WS-CAF specifications also include an innovative approach to transactional interoperability, spanning multiple models, multiple middleware, and multiple devices when they are included within asynchronous business process flows.

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.

Introduction

The Web Services Composite Application Framework (WS-CAF) is divided into three parts:

- Web Service Context (WS-CTX), a lightweight framework for simple context management
- Web Service Coordination Framework (WS-CF), a sharable mechanism to manage context augmentation and lifecycle, and guarantee message delivery
- Web Services Transaction Management (WS-TXM), comprising three distinct protocols for interoperability across multiple transaction managers and supporting multiple transaction models (two phase commit, long running actions, and business process flows)

The overall aim of the combination of the parts of WS-CAF is to support various transaction processing models and architectures. It is important to realise also that the individual parts of WS-CAF are designed to complement Web services orchestration and choreography technologies such as BPEL [1],WSCI [2] and WS-Choreography [3] and work with existing Web services specifications such as WS-Security [4] and WS-Reliability [5]. The parts define incremental layers of functionality that can be implemented and used separately by these and other specifications separately or together. The emphasis of WS-CAF is to define supporting services required by Web services used in combination.

Furthermore, the parts of WS-CAF comprise a stack, starting from WS-CTX, adding WS-CF, and finally WS-TXM to deliver the complete features and functionality required by composite applications. An implementation of WS-CAF can start with WS-CTX for simple context management, and later add WS-CF for its additional context management features and message delivery guarantees, and finally add WS-TXM for managing a variety of recovery protocols. Similarly, a composite application can use the level of support required, from simple context management through to transactional recovery mechanisms.

Relationship of the specifications

In Figure 1 we can see the relationship between the various specifications and the protocols they support. This shows the hierarchy of transaction protocols currently supported by WS-TXM.

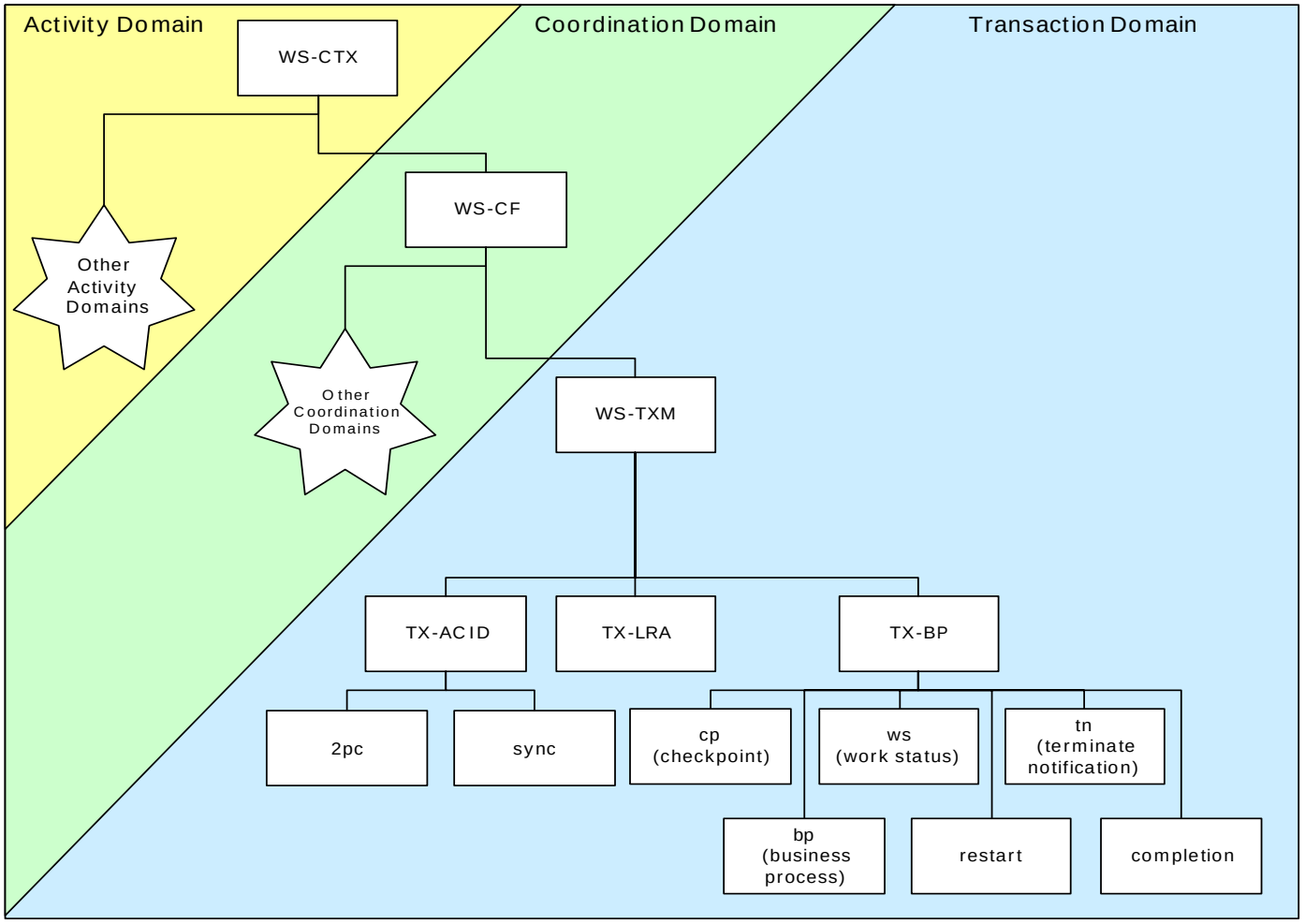

Figure 1, Relationship between specifications and transaction protocols.

WS-CAF concepts are based on the assumption that multiple Web services are often placed into various relationships to accomplish a common purpose and therefore at a minimum need a way to share common context (the Activity Domain), and at a maximum need a way to coordinate results (the Coordination Domain) into a single, potentially long-running larger unit of work with predictable results despite failure conditions (the Transaction Domain).

Web services typically cooperate to perform a shared function, such as multiple related operations on a shared resource such as a database or display, or processing different portions of a purchase order using a predefined sequence, and may require one or more of these domain services depending on the shared function.

Definition of Terms

Cooperating Web services are called *participants* in a composite unit of work. Participants are minimally identified as Web services that share a common context. A *context* is a data structure containing information pertinent to the shared purpose of the participants, such as the identification of a shared resource, collection of results, common security information, or pointer to the last-known stable state of a business process.

One of the main benefits of sharing context is that all participants can discover the result of the execution of the overall cooperating unit of work, or *scope* of the shared function. In many cases, depending on the success or failure of one or more of the participants, other participants may want to be notified or even directed to perform specific actions. Tracking execution results (i.e. success or failure), notification of results, and ensuring a common outcome upon failure are other ways to describe the WS-CTX, WS-CF, and WS-TXM layers.

One of the main benefits of a *coordinator* is that it can take the responsibility for notifying the participants of the outcome, persisting the outcomes of the participants, and managing the context. The *outcome* summarizes the results obtained by executing the cooperating Web services, and outcome-based behaviors are be defined using various protocols.

A coordinator becomes a participant when it registers itself with another coordinator for the purpose of representing a set of other, typically local participants. When a coordinator represents a set of local participants, this is called *interposition.* Interposition assists in achieving interoperability because the interposed coordinator can also translate a neutral outcome protocol into a platform specific protocol.

The main benefit of adding *transaction*-based protocols is that the participants and the coordinator negotiate a set of agreed actions or behaviors based on the outcome, such as rollback, compensation, synchpoint, three-phase commit, etc.

Usage Models

Using the basic context service, composite Web services can manage and interpret shared context themselves, or optionally, depend upon a shared Web service with which they each interact to manage and interpret the context. Using the Coordination Framework, Web services register with an external service that takes responsibility for managing and interpreting the shared context, and coordinating the notification messages sent to participants upon completion of the shared work. Using one or more of the pluggable transaction models, the coordinator can coordinate completion activities depending upon the outcome of the shared work and the selected transaction model.

WS-TXM defines three pluggable transaction models:

- Basic (familiar two-phase commit) with interoperability across multiple transaction managers.
- Extended (including asynchronous coordination and compensation) for long running and asynchronous transactions.
- Business process, for treating all steps in an automated business process as part of a single logical transaction, including occasionally connected computing.

These transaction models can be used separately or in combination to support a wide range of business transactions from small, quick updates to long running business processes, including a variety of mobile devices.

Comparison with other specifications

Other specifications have been published in the area of Web services transactions. In contrast to the other specifications, the WS-CAF:

- Provides a basic mechanism for context sharing and defines the context as a Web resource that can be independently implemented
- Defines a neutral, abstract transaction protocol for mapping to existing implementations
- Models transactions at three levels of abstraction to fit different types of business transactions
- Layers technology appropriately from basic to complex functionality for ease of implementation and use, allowing implementations to start simple and move to complex as appropriate and necessary
- Achieves broad interoperability through the use of neutral protocols and interposition

The WS-CAF can use any transaction protocol[1] in place of or in addition to the neutral protocols defined in WS-TXM. WS-CAF does not require the implementation of a new transaction protocol. Rather, WS-CAF decomposes the problem into appropriately sized layers and allows implementations to implement part or the entire framework.

Positioning the parts

The three parts of WS-CAF comprise a stack of functionality, supporting transaction processing concepts and capabilities from the simple to the complex. For example, each participant in a basic composite application with shared context has the ability to

[1] Such as WS-T [6] or BTP [7].

discover the results of the other participants' execution and take appropriate action. Adding the coordinator guarantees that participants will be notified, and adding a transaction protocol provides a means by which the participants can negotiate with each other to take coordinated actions as the result of knowing a common outcome. In particular:

- The WS-CTX specification defines a very simple mechanism that allows multiple Web services to share a common context structure, which is designed to be used independently of the WS-CF and WS-TXM specifications. WS-CTX ensures that all Web services participating in a composite application share a common context and can exchange information about a common outcome.
- The WS-CF specification defines a coordinator that provides additional features for persisting context operations and guaranteeing the notification of outcome messages to the participants. WS-CF is designed to complement WS-CTX and can be used independently of the WS-TXM specification.
- The WS-TXM specification defines transaction protocols for ensuring that a common outcome is negotiated with all participants in a composite application. While WS-CF is responsible only for notifying the participants of the outcome, WS-TXM defines a protocol for the participants to coordinate outcomes with each other and make a common decision about how to behave, especially in the case of failure. WS-TXM is designed to ensure that any given composite application always either reaches successful completion, or reverts to a predictable, known state in the failure case of one or more of the individual Web services in a composite application such as a business process or choreography.

The Context Service

Figure 2 illustrates WS-CTX as comprising one or more Web services participants that share a common context.

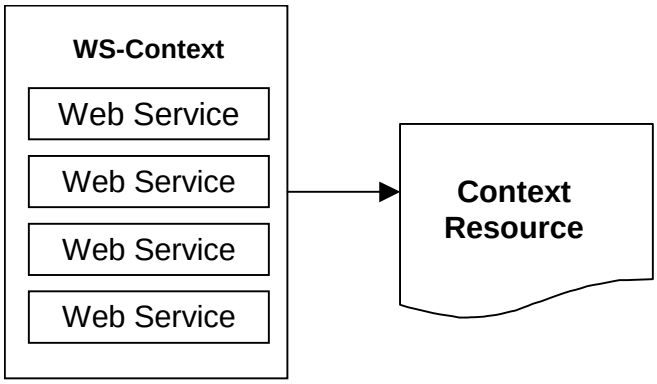

Figure 2, Context Service.

The context is modeled as a Web resource and is accessible via a URI. Web services are identified as participants in the activity by including the context URI in the SOAP header, as shown in the following example:

```
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-
envelope">
 <env:Header>
 <n:Composite
xmlns:n=”http://example.org/CompositeApplication”>
  <n:Context>
     http://example.org/contextURI
  </Context>
 </n:Composite>
 </env:Header>
 <env:Body>
 <m:Message xmlns:m=”http://example.org/MessageSchema”
  <m:…
 </m:Message>
 </env:Body>
</env:Envelope>
```

Example 1, Including the context URI in the SOAP header.

Web services can also choose to join a composite application upon receipt of a SOAP message containing the context URI in the header, or, optionally, containing the context itself within the body of the SOAP message. In this way the context service supports passing the context by reference (a URI in the header) or by value (within the SOAP body).

A Context Service is modeled as a Web service that can be co-located with the participants or executed as a separate service. All Web services referencing the same context URI, or accepting the same context in a SOAP message, are considered part of

the same *composite application*. An implementation may permit the Web services to manage the context themselves, or use a separate service to manage the context. Context is defined as a Web resource, accessible via dereferencing the URI and can be passed by reference as a URI in a SOAP header or by value within the SOAP body.

Context

The purpose of a context is to allow multiple individual Web services to enter a relationship by sharing certain common attributes as an externally modeled entity. Typical reasons for Web services to share context include:

- Common security environment – multiple Web services execute within a single authentication session or authorization check
- Common outcome negotiation – multiple Web services or activities execute within a single transactional unit of work whose outcome must be coordinated
- Common access to such resources as database management systems, files, queues, and displays such that multiple Web services can share a connection
- Participation in an automated business process execution, or choreography

The definition of a context is application specific, but contains at a minimum a unique ID in the form of a namespace URI. The namespace URI qualifies the elements in context, which can include any or all of the following:

- Security tokens or attributes
- Transaction IDs
- File ID
- Database session ID
- Display ID
- Results for each individual execution
- Business process automation context (i.e. document or schema)

The outcome of a coordinated activity or transaction is managed either by the context service (using lightweight mechanisms that do not guarantee notification or persistence of the results) or by adding the coordinator (which does guarantee notification and persistence of the results).

A basic composite application is distinguished from a transactional composite application by the latter's ability to negotiate a recovery action in the event of failure. Failure is defined as one or more of the participants returning a fault that cannot be handled by the application. A basic composite application has no mechanism for negotiating recovery actions in the case of failure.

Scoping

The ability to scope arbitrary units of distributed work by sharing common context is a requirement in a variety of distributed applications, such as choreography and business-to-business interactions. Scoping makes it possible for Web services participants to be able to determine unambiguously whether or not they are in the same composite application, and what it means to share context. Scopes can be nested to arbitrary levels to better delineate application work.

The context contains information, including a unique context identifier (URI), the URI of the initial requester, the URI of the ultimate provider, optional URIs of each additional Web service within the scope of the composite application, optional security information, and a status result for each Web service in the composite application.

A composite application is defined as a collection of Web services that executes in a specified sequence for the purpose of carrying out multiple operations on a shared resource, such as a database, display, or XML document, in which the unresolved failure of an individual Web service execution causes the failure of the entire application. Any Web service execution that fails has the responsibility to notify any previously successful Web services of the failure, and terminate the application. A failure can be resolved if the service requester includes sufficient failure recovery logic.

A Web service is identified as belonging to a composite application by the inclusion of the context URI in the header and optionally a namespace URI identifying the list of Web services participating in the application. The namespace URI can be translated in a variety of ways, including referencing an XML document containing the list or a WSDL providing the interface to the list of participants. In the case where Web services register themselves with a coordinator, the coordinator maintains the list and the namespace URI is interpreted as referencing the coordinator location.

For example:

```
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-
envelope">
 <env:Header>
 <n:Composite
xmlns:n=”http://example.org/CompositeApplication”>
  <n:Context>
      http://example.org/contextURI
  </n:Context>
  <n:ApplicationList>
      http://example.org/ListLocation
  </n:ApplicationList>
 </n:Composite>
 </env:Header>
 <env:Body>
 <m:Message xmlns:m=”http://example.org/MessageSchema”
```

```
  <m:…
 </m:Message>
 </env:Body>
</env:Envelope>
```

Example 2, Identifying the participants in a composite application.

The shared context allows a series of operations to share a common outcome. The Web services participating in the composite application maintain the context information shared between multiple participants in a Web services interaction.

The first Web service invoked within a composite application using WS-CTX only has the responsibility for initializing the context and updating the context with its execution result, unless the result is failure, in which case the composite application is cancelled. Each subsequent Web service invoked within an application has the responsibility for updating the context with its execution result unless the result is failure, in which case the application is cancelled, and any Web service execution prior to the current one is notified of the failure.

Notification in a composite application using WS-CTX is accomplished using a best effort. Each Web service in the application is responsible for invoking any recovery or compensation logic. When a WS-CF defined coordinator is added to the picture, the coordinator takes responsibility for notifying each participant. The participants remain responsible for invoking any recovery or compensation logic, however. When a WS-TXM defined transaction protocol is added, the coordinator and the participants share responsibility for executing the recovery or compensation activities defined within the transaction protocol.

Transport issues aside, an important point is that context information can be specific to the type of application being executed, such as common operations on a shared resource, identifying a transaction coordinator, identifying the other participants in an application, or propagating recovery information in the event of a failure.

A single context type is not sufficient for all types of composite applications. Hence, the context must be extensible in an application specific manner.

The Coordination Framework Service

Figure 3 illustrates how individual Web services as well as composite applications can register as participants with a coordinator, which takes over responsibility for context management and notifying participants of the outcome of a series of related Web services executions. The executions are defined as related when they share a common context. As the illustration shows, a coordinator can register itself with another coordinator and become a participant, thereby improving interoperability. Individual Web services can

also register themselves with the coordinator and therefore join the composite unit of work.

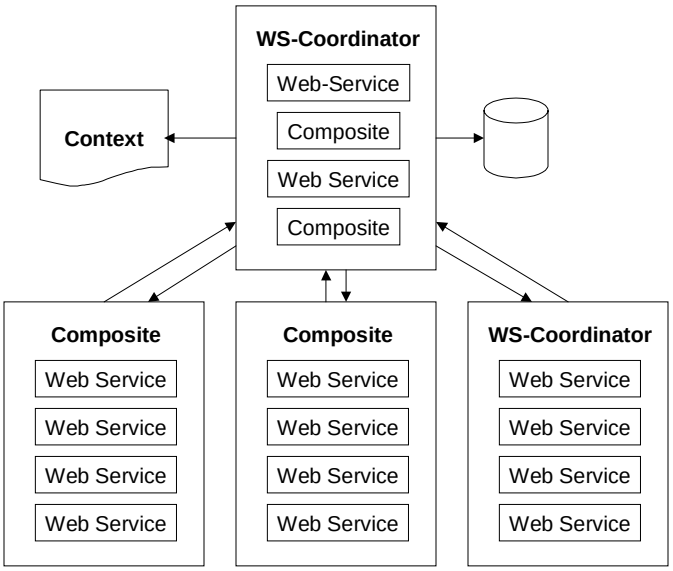

Figure 3, Relationship of coordinator to Web services and composite applications.

The following example illustrates the use of a coordinator URI reference in the SOAP header:

```
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-
envelope">
 <env:Header>
 <n:Composite
xmlns:n=”http://example.org/CompositeApplication”>
  <n:Context>
      http://example.org/contextURI
  </n:Context>
  <n:Coordinator>
      http://example.org/coordinatorURI
  </n:Coordinator
 </n:Composite>
 </env:Header>
 <env:Body>
 <m:Message xmlns:m=”http://example.org/MessageSchema”
  <m:…
 </m:Message>
 </env:Body>
</env:Envelope>
```

Example 3, Adding the coordinator URI reference to the message.

As shown in Example 3, including a coordinator URI in a SOAP header identifies to the SOAP processor that the Web service is registering with the coordinator defined by the WSDL interface returned by dereferencing the URI.

The Transaction Model Service

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 that 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.

Figure 4 illustrates the layering of WS-TXM protocols onto the coordinator. WS-TXM defines a set of pluggable transaction protocols 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 (scopes). As mentioned previously, scopes can be nested (parent-child relationships) and concurrent, representing application tasks. Counter-effects for completed scopes may then become the responsibility for the enclosing scope. Examples of coordinated outcomes include the classic two-phase commit protocol, long running outcomes, open nested transaction protocol, asynchronous messaging protocol, or business process automation protocol.

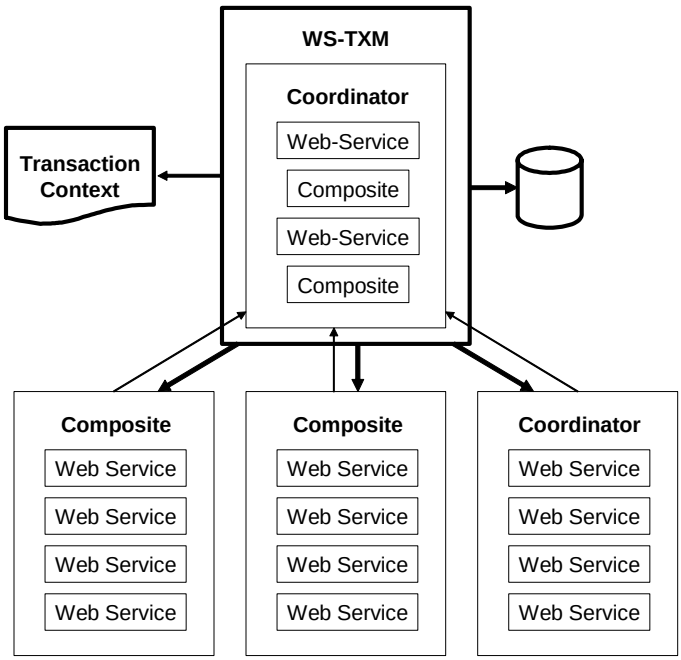

Figure 4, Relationship of transactions to coordination framework.

Coordinators can be participants of other coordinators, as described above. When a coordinator registers itself with another coordinator, it can represent a series of other Web services for the purpose of mapping a neutral transaction protocol onto a platform-specific transaction protocol.

A composite application can be a business process, or a step within a business process.

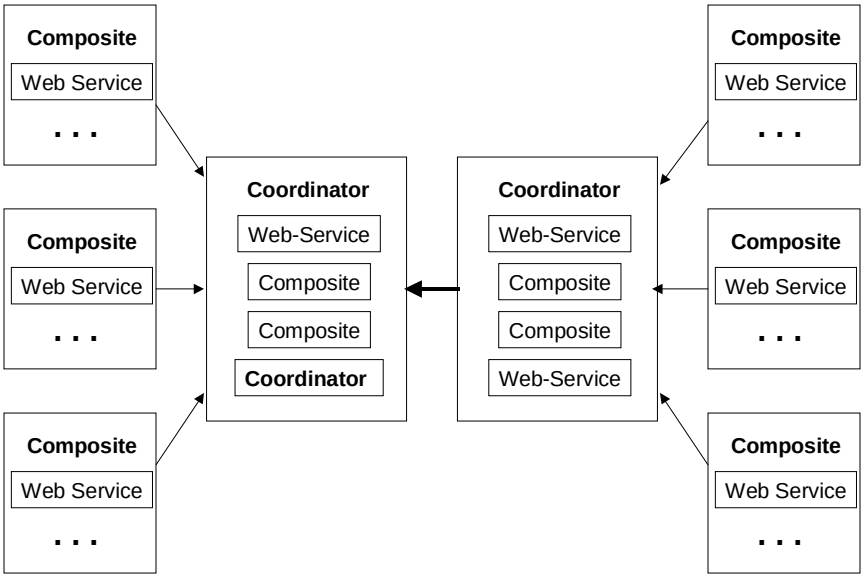

Figure 5, Interposition architecture.

As shown in Figure 5, a coordinator can register itself as a participant to another coordinator. This architecture is called interposition, and allows a coordinator to map represent a series of local activities as a single unit, coordinating the local composite application and sending and receiving single messages with the other coordinator.

There are three transaction protocols defined by WS-TXM:

- *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.

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 application is required to be compensatable. Therefore, an application's work is either performed successfully or undone. How individual Web 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.

In the LRA model, each application 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 service informs the individual service(s) that it is no longer required.

Let's consider another example of a long running business transaction. The application is concerned with booking a taxi, reserving a table at a restaurant, reserving a seat at the theatre, and then booking a room at a hotel. If all of these operations were performed as a single transaction then resources acquired during booking the taxi (for example) would not be released until the top-level transaction has terminated. If subsequent activities do not require those resources, then they will be needlessly unavailable to other clients.

Figure 6 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.

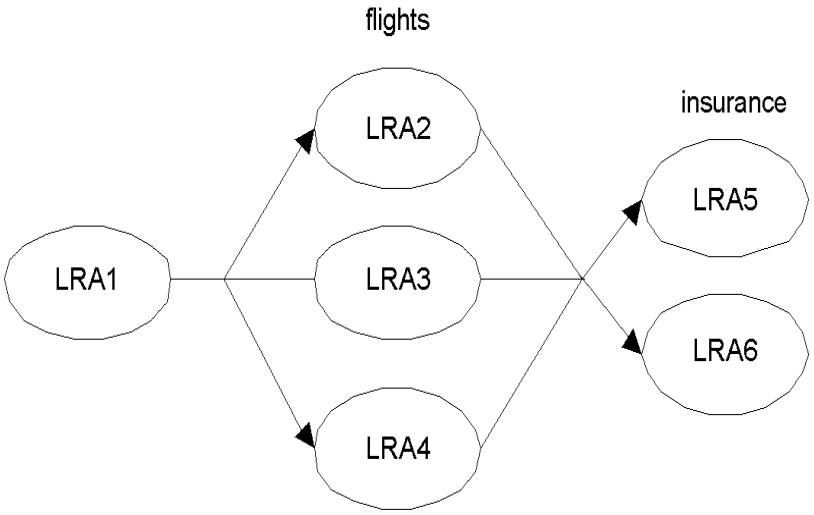

Figure 6, LRA example.

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.

Each domain may represent a different transaction model if such a federation of models is more appropriate to the activity. Each business task (which may be modeled as a scope) may provide implementation specific counter-effects in the event that the enclosing scope must cancel. In addition, periodically the controlling application may request that all of the business domains checkpoint their state such that they can either be consistently rolled back to that checkpoint by the application, or restarted from the checkpoint in the event of a failure.

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 7. 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 modeled as a separate task, or Web service. In this example, the HiFi task is actually composed of two sub-tasks.

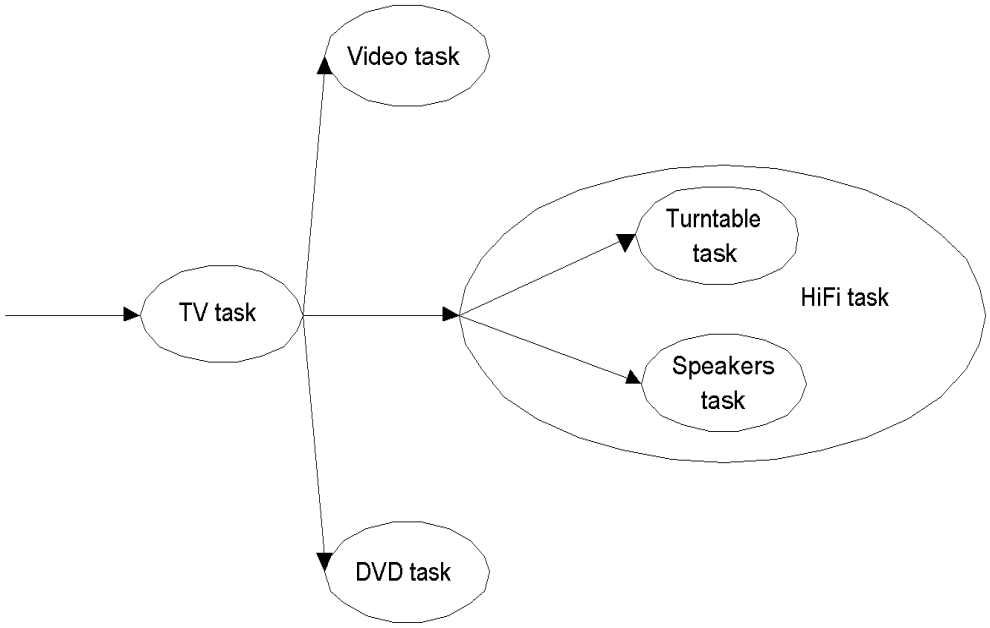

Figure 7, Business processes and tasks.

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).

Conclusions

This document has outlined a solution to the general requirement for predictable, dependable *business-to-business transactions*. We have shown that a single solution to this problem is unlikely to occur. In fact, we have shown that, business-to-business transactions are a subset of *business-to-business interactions*. Context management and coordination are more fundamental requirements than transactions and therefore should be considered as separate standardization points.

Context manipulation by itself is useful in business interactions purely from the perspective of correlation and auditing. Coordination is used in a variety of services, including security and transactions. In both cases, requiring each domain in which context or coordination is used to implement their own non-standardized solutions is inefficient and affects interoperability. Therefore, just as there is a requirement for standardizing on higher-level services such as transactions or security, it is important for us to standardize on any component (or sub-service) infrastructures they may require.

We have shown how an architecture based on a hierarchy of context, coordination and transactions can be created and provides a useful separation of concerns whilst at the same time offering flexibility of implementation at each level. The different transaction models proposed by WS-TXM address different problem domains in order to provide efficient, targeted solutions. Trying to provide a single solution that fits all possible use cases is not possible for a number of reasons, not least of which is that Web services is a new and evolving area: today's problem domains are not necessarily tomorrow's. Therefore, flexibility and extensibility are key to any proposed solution.

References

[1] Business Process Execution Language for Web Services, version 1.1, http://www.oasis-open.org/committees/download.php/2046/BPEL%20V1-1%20May%205%202003%20Final.pdf

[2] Web Services Choreography Interface (WSCI), August 2002, http://www.w3.org/TR/wsci/

[3] Web Services Choreography Working Group, http://www.w3.org/2002/ws/chor/

[4] Web Services Security specification, http://www-106.ibm.com/developerworks/library/ws-secure/

[5] Web Services Reliable Messaging, http://www.oasis-open.org/committees/download.php/1461/WS-ReliabilityV1.0Public.zip

[6] Web Services Transactions Specification, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-transaction.asp, August 2002.

[7] BTP Committee specification, http://www.oasis-open.org/committees/business-transactions/, April 2002.

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