WS-CAF: Contexts, Coordination and Transactions for Web Services

Mark Little[1], Eric Newcomer[2] and Greg Pavlik[3].
[1] *Arjuna Technologies Ltd,*
[2] *IONA Technologies Ltd,*
[3] *Oracle Corporation*

Abstract

There is considerable discussion in the Web services community about how to create composite applications for business process automation. Until recently no one has been focusing on a comprehensive solution to the difficult system-level problems that arise when Web services are combined. The OASIS Web Services-Composite Application Framework (WS-CAF) specifications provide the means to solve common infrastructure related problems found in typical composite applications such as order entry, inventory management and distributed transaction support.

As Web services have evolved as a means to integrate processes and applications at an inter-enterprise level, traditional transaction semantics and protocols have proven to be inappropriate. Web services-based transactions, colloquially termed Business 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.

A solution to this problem has to work over HTTP and include existing transaction processing technologies of all types: database management systems, application servers, message queuing systems and packaged applications. The solution needs to support a range of requirements including lightweight applications in which the major goal is to let Web services know when they're in the same application to complicated transactions that may take days or weeks to complete across wide ranging geographies, time zones, and enterprise boundaries. In this paper we'll look at the WS-CAF standardization effort and show how it is attempting to address this important and difficult subject. We'll also consider how the architecture defined by WS-CAF fits into the evolving architecture of Web services and give an indication of where we think things are going in the future.

1. Introduction

It is often said that Web services are immature and missing some features compared to other distributed computing development environments such as CORBA, DCOM, and J2EE, but exactly what needs to be added is the subject of considerable debate. Many proposals have surfaced in the form of Web services specification drafts and as the number of proposals and specifications grows, confusion often grows rather than shrinks.

However, a common foundation exists underneath many of these missing pieces: *context management* (essentially the ability to associate disparate entities within the same unit of distributed work). This fact, and the problem that no such facility exists in Web services, came to light soon after SOAP 1.1 was submitted to W3C, when we first started work on mapping the Transaction Internet Protocol (TIP) to SOAP back in mid-2000. Transactioning is often mentioned as one of the major features for distributed computing environments and CORBA, DCOM/.NET, and J2EE all provide it, so it is a fairly obvious requirement for Web services. However, we quickly realized we had a larger problem than simply adding a transaction context to the SOAP header and we had to suspend the effort.

TIP was dependent on a *session-oriented communication protocol* for exchanging two-phase commit commands. Like most distributed transaction protocols, TIP required a persistent transaction context to be shared among the communicating parties in the transactional operation so that a two-phase commit protocol can be executed reliably. An abort (or rollback) can be triggered automatically when a communication connection is dropped. It is too risky to the health of the resource managers being coordinated not to rollback when communication is lost, but without a persistent session mechanism, the client (the transaction root) is unable to detect connection loss.

Unfortunately this type of behavior is in fact impossible to define for a communication system based on HTTP, where sessions are maintained only long enough to transfer an HTML page and are dropped immediately afterward. This behavior is tremendously helpful to support a system of the scale of the World Wide Web, but it is not so helpful when you need to support a classic transactioning protocol such as two-phase commit.

Many other typical features and functions of distributed systems also depend upon persistent sessions, including secure sessions, conversations, and load balancing and failover mechanisms. The way to think about the use of what we are calling *persistent sessions* in general is the ability to get back to the same place where you left off in a remote program on a subsequent call. In the specific cases of transactions, or secure conversations, for example, this is the ability to maintain the context of the first operation while waiting for the next to arrive.

As we shall see in the following sections, the OASIS Web Services Composite Application Framework [2] attempts to solve this problem by defining core support in the Web services architecture for context management. It also builds upon this to provide the necessary transaction functionality that we were unable to accomplish back in 2000.

1.1 An overview of WS-CAF

In general, *composite applications* are increasing in importance as companies combine off-the-shelf and homegrown Web services into new applications. Various mechanisms are being proposed and delivered to market daily to help improve this process. New "fourth generation" language development tools are emerging that are specifically designed to stitch together Web services from any source, regardless of the underlying implementation.

A large number of vendors are starting to sell business process management, workflow and orchestration tools for use in combining Web services into automatic business process execution flows. In addition, a growing number of businesses find themselves creating new applications by combining their own Web services with Web services available from the Internet supplied by the likes of Amazon.com and Google.com.

These types of composite applications represent a variety of requirements, from needing a simple way to share persistent data to the ability to manage recovery scenarios that include various types of transactional software. Composite applications therefore represent a significant challenge for Web services standards since they are intended to handle complex, potentially long-running interactions among multiple Web services as well as simple and short-lived interactions.

The WS-CAF suite includes three specifications that can be implemented incrementally to address the range of requirements needed to support a variety of simple to complex composite applications:

- Web Service Context (WS-CTX), a lightweight framework for simple context management.
- Web Service Coordination Framework (WS-CF), which defines the behavior of a coordinator with which Web services can register for context augmentation and results propagation, and on top of which can be plugged various transaction protocols.
- 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 or compensation, and business process flows).

The overall aim of the combination of the parts of WS-CAF is to provide a complete solution that supports various transaction processing models and architectures. Implementations of WS-CAF can start small and grow to include more functionality over time. WS-CAF specifications are designed to compliment Web services orchestration and choreography technologies such as WS-BPEL [3] and WSCI [4] and are compatible with other Web services specifications. The emphasis of WS-CAF is to define supporting services required by Web services used in combination, including other specifications.

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 context message delivery guarantees, and finally add WS-TXM for managing a variety of recovery protocols.

In the following sections we shall examine each of these specifications in more detail and show how they support the development of composite Web Service applications.

1.2 Context management

WS-Context provides a mechanism for Web services to share persistent state, which is required to support conversational interactions, single sign-on, transaction coordination, and other features dependent upon system-level data items such as IDs, tokens etc. Context provides a way to correlate a set of messages into a larger unit of work by sharing common information such as a security token exchanged within a single sign on session.

Because distributed computing systems depend upon a variety of IDs, tokens, channels, and addresses, which are a part of every software infrastructure, and because Web services are independent of any particular execution environment, this type of system level information needs to be organized and managed in a persistent, shared context structure. Applications need a service to manage the lifecycle of the shared context, and to ensure the context structure is kept up to date and accessible.

Through the use of shared context Web services from different sources can effectively become part of the same application because they share common system information. A classic example is a single sign-on mechanism that allows a user or an application to present authentication credentials to access to a set of cooperating Web services. Application level context, such as a shared document, can also benefit from a generic context management service.

WS-Context defines a context data structure that can be arbitrarily augmented. By default, all the context defines is a unique context identifier, the type of the context (e.g., transaction or security) and a timeout value (how long the context can remain valid). Like SOAP headers, which WS-

Context can replace or combine for easier management, the context data structure includes an attribute requiring the context to be understood and/or propagated. For example:

```
<ContextType> MyContext </ContextType>
  <context-identifier>

www.webservicestransactions.org/example/Middlewar
e2004ContextExample
  </context-identifier>
. . .
. . . mustUnderstand=true
. . . mustPropagate=true
. . .
  <child-contexts>
    <child-context>
      <user-name> EricNewcomer </user-name>
      <password> ******** </password>
    </child-context>
    <child-context>
      <database-name> SQL-DB </database-name>
      <file-name> Index-S-file </file-name>
      <display-address> PocketPc25 </display-
address>
    </child-context>
    <child-context>
      <transaction-type> BusinessProcess
</transaction-type>
      <transaction-mode> Required </transaction-
mode>
    </child-context>
  </child-contexts>
```

The context structure shown above includes "children" that can be used to share information needed to process a request on behalf of the user of a composite Web service. In this case, the context includes the `mustUnderstand` attribute set to `true` to indicate that the context must be understood in order to process the request, since it contains information necessary for successful completion of the request. The context has also been marked as `mustPropagate=true`, meaning that each Web service in the composite must receive or be able to access the context to ensure proper execution.

The example illustrates user information that obtains a security token and passes the token as a single sign-on feature for the composite application. In other words, the context could be provided as input to the first Web service in a WS-BPEL defined flow. The first Web service in the flow then could check the username and password (the asterisks are used to indicate opaque data in the example) and retrieve an authentication token to use in checking whether the user is authorized to access each subsequent Web service in the flow. Such an authentication token would be placed back into the context data structure as an augmentation to the original structure. For example:

```
    <child-context>
      <user-name> EricNewcomer </user-name>
      <password> ******** </password>
      <AuthToken> ******** </AuthToken>
    </child-context>
```

The `AuthToken` is added by the security system at the end of the username and password information upon execution of the initial Web service in the flow. The context is a living data structure; the results of a security sign on (or other operation pertinent to the contents of the context) would typically be added for propagation to the next Web service in the flow. For example, a single sign on system bridging multiple security domains would add another token to the context.

1.3 Coordination

A *coordinator* is a software entity responsible for ensuring consensus is achieved between multiple parties. Coordinators exist in CORBA, .NET, J2EE, and other distributed computing environments to coordinate the classic two-phase commit transaction protocol across multiple data resources. However, coordination is a more fundamental requirement: it is used in security, replication, caching and other areas.

Therefore, the definition of a coordinator in WS-CAF is extended for use with Web services by using a plug in mechanism that supports multiple coordination protocols such as the classic two-phase commit protocol, long running actions with compensation, and complex business process and orchestration flows.

Web services are designed to be multi-protocol and therefore to map to multiple underlying technologies. Instead of tying the coordinator to the two-phase commit protocol, which is the way current coordinators are defined, the WS-CF specification creates a general-purpose coordinator capable of driving a variety of context types and transaction protocols (such as those defined in WS-TXM and others).

```
<env:Envelope
xmlns:env="http://www.w3.org/2002/12/soap-
envelope">
  <env:Header>
    <n:Composite
xmlns:n="http://example.org/CompositeApplication"
>
      <n:Coordinator>

http://www.webservicestransactions.org/example/Co
ordinatorURI
      </Cooordinator>
    </n:Composite>
  </env:Header>
  <env:Body> ...
```

```
</env:Envelope>
```

In the above example, the coordinator URI points to a Web service interface that defines the SOAP message pattern for interactions between the coordinator and the Web service execution. The coordinator then manages any user defined context and generates and propagates any context for use within the operations of the composite and includes each registered Web service in the recovery protocol. When multiple Web services register with the coordinator to use the same context type, the message exchange pattern includes all Web service executions within the composite. In other words, the scope for a given context type is determined by the Web services that register with the coordinator to share it.

The message exchange pattern described for the Web services in the application isn't changed. By registering with the coordinator, however, a separate message exchange pattern is established as a secondary, system-level interaction to handle the context propagation and recovery operations. The two message exchange patterns are linked using the context ID passed in the SOAP header and given to the coordinator upon registration.

1.4 Transactions

Transaction processing is at the core of business. Every exchange of money for goods is a transaction, as are most other activities within commerce, military, and science. Transaction processing technology ensures that any activity's operations on data are recorded consistently on computer systems, so that the systems remain as reliable indicators of the "real world" as their paper-based antecedents did, or at least as closely as possible given the vagaries of the electronic medium.

For example, a bookstore's database must always accurately reflect in store inventory of what's on the shelves. Manual processes to confirm inventory are costly, as are errors in bank transfers, telephone billing, and manufacturing line preparation. Transaction processing technology helps all of these now automated activities run smoothly and as expected, despite system failure, which can be counted upon to regularly occur. Computers, being basically unstable electronic devices, are subject to all manner of problems.

Web services are loosely coupled XML interfaces to computer systems comprised of programs, objects, and databases. Web services are used to solve a variety of interoperability and integration problems. Because many of these existing systems are concerned with processing transactions, it is critical to appropriately and correctly include transaction-processing (TP) information within Web services that are integrating TP systems, allowing Web services to participate in a critical area.

However, up to this point in the evolution of Web services standards, several attempts have been made to meet the requirement for defining transactions for Web services, but nothing as of yet has provided the complete solution. WS-CAF is as close as anything has come to this to date. Although some work remains to complete the specifications, the initial work provides a very solid foundation.

Two-phase commit is the most common distributed transaction protocol in use today. While two-phase commit is insufficient for long-running, widely-distributed Web services flows, the protocol is well understood and in particular its requirement for an independent coordinator lays the foundation for some of the extended models such as compensations and business process transactions that are needed for Web services.

As such, the WS-TXM specification defines three transaction models that can be plugged into WS-CF, including a two-phase commit protocol. However, the two-phase commit protocol in WS-TXM is designed specifically to support interoperability across multiple variations of the two-phase commit protocol that exist in current and proposed systems.

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 1 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.

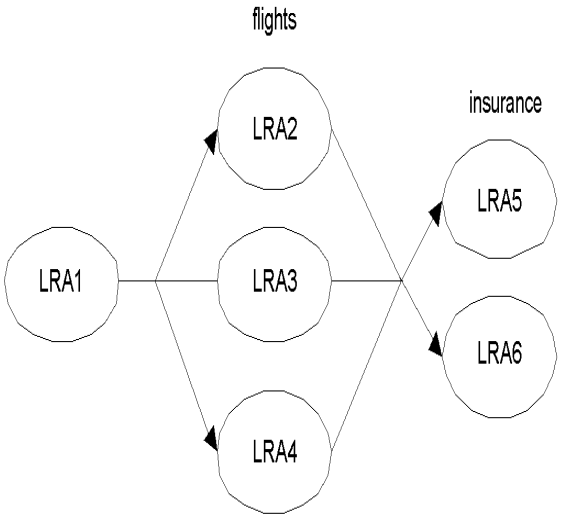

Figure 1, LRA example.

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 2. 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.

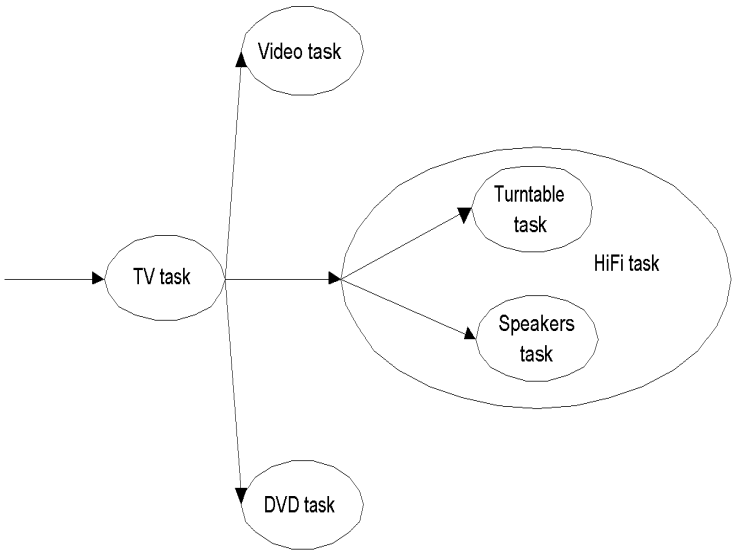

Figure 2, Business processes and tasks.

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).

2. Comparison with other specifications

The WS-CAF specifications are designed to work with and complement other Web services specifications, including WS-Security, WS-Reliability, WS-BPEL, and others. The WS-CAF specifications define the SOAP message exchange patterns and WSDL interfaces necessary to accomplish the context management, coordination, and transaction processing capabilities needed to support composite application executions.

The question of compatibility with other Web services specifications is a difficult one since so many specifications are under progression at various standards bodies and through private consortia. It's often hard to know where any particular Web services specification fits within the overall picture. The W3C is producing a Web Services Architecture specification on this topic [5], while IBM and Microsoft have produced a whitepaper to reflect their own view [6]. At this point in time neither seems definitive, which is understandable given the rate of

change still occurring in Web services and the fact that no single standards body is in control, and that so many specifications remain under private copyright.

An important consideration with respect to Web services specifications is the issue of intellectual property rights and copyright ownership. The WS-Interoperability organization [7] for example has debated to what extent their profiles can or should reference private specifications. The WS-I Basic Profile references SOAP 1.1, WSDL 1.1, and UDDI V2, all of which were produced by private consortia but have since been submitted to a standards body.

Some specifications under private copyright ownership require royalty fees to be paid to the copyright owners for the right to implement and sell software based upon them. Web services vendors who are not copyright holders on a given specification may be concerned about implementing a specification that their competitors control, especially when they are not allowed to participate in its definition or evolution. When a specification is not under the control of a single vendor of group of vendors, it's said to be "open," meaning that anyone can participate in its definition and evolution.

With respect to other Web services specifications, both private and open, the WS-Context specification is unique. No other specification exists that defines a generic context management mechanism for Web services.

The OASIS WS-Coordination Framework specification shares a common derivation with the private WS-Coordination specification – both are based on the Object Management Group's (OMG) extended transaction specification called *Additional Structuring Mechanisms for the OTS* [8]. This specification was developed as an extension of the Object Transaction Specification (OTS) [9], which defines how coordination works for both the CORBA and J2EE worlds.

The *Additional Structuring Mechanisms* specification, sometimes called the *Activity Specification* since it defines generic activities, pioneered the concept of a pluggable coordinator. The specification includes an example of an open nested transaction model to validate the design of a coordinator as a generic state machine capable of supporting multiple transaction protocols, rather than tying the coordinator to the two-phase commit protocol (as it is in the base OTS specification). WS-CF, like WS-C, is derived from this pioneering OMG work.

The base OTS specification also contains a precedent for WS-CAF because it defines how multiple coordinators can work together. The concept is called *interposition*, and it means that a coordinator can act as a resource to another coordinator on behalf of a set of local resources. The idea was included in the OTS specification as a network optimization, but it turns out to be useful for interoperability as well.

In IONA's Orbix Mainframe product [10], for example, an interposed coordinator bridges the standard OTS two-phase commit protocol from a CORBA object or EJB on Unix or Windows to the proprietary Resource Recovery Management Services (RRMS) two-phase commit protocol on the mainframe. Bi-directional transactional interoperability with CICS and IMS is achieved using an interposed coordinator on the mainframe to map the standard OTS two-phase commit commands into and out of their RRMS equivalents. The standard OTS protocol is used over the wire.

3. Conclusions

Specifications such as the Business Process Execution Language (BPEL) and the Web Services Choreography Interface (WSCI), focus on tying multiple Web services together to create multi-step applications, such as filling a purchase order or resolving an insurance claim. Therefore these applications have the requirement to share context across the steps.

The WS-CAF specifications define a standard framework for use by a set of cooperating Web services so that:

- Each Web service knows what application it's included in (or how many and which one it's currently in).
- The Web services in a composite have a way to obtain results of another Web service's operations.
- A standard mechanism is available to share needed system data such as security tokens, file and device handles, or network addresses.
- The application can set rules and policies for recovering from the failure of one or more of the services.

While specifications such as BPEL and WSCI provide the mechanism for extending the WSDL layer to identify a series or sequence of execution for multiple Web services, WS-CAF defines the complementary system layer necessary to ensure that the multiple Web services achieve the desired results of the application, and that the cooperation of multiple Web services from whatever source (local or remote) produces predictable behavior despite system failure and leaves the system in a known state.

As with most aspects of standardization, the value in WS-CAF is derived from the potential for its features and functions to be provided by Web services vendors, therefore helping application developers solve composite application problems more easily. Once adopted and implemented, the functionality contained within WS-CAF will not only be available as part of the platform (and therefore not have to be coded as part of the application) but also it will be available in a standard way across platforms, allowing Web services from multiple

environments to interoperate more easily, efficiently, and effectively than if the developers had to code all of the equivalent features and functionality themselves in a non-standard way.

4. References

[1] RFC 237, 1998 http://www.faqs.org/rfcs/rfc2371.html

[2] OASIS Web Services Composite Application Framework Technical Committee, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-caf

[3] OASIS Web Services Business Process Execution Language Technical Committee, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

[4] The Web Services Choreography Interface, August 2002, W3C Note, http://www.w3.org/TR/2002/NOTE-wsci-20020808/

[5] W3C Architecture Committee, see http://www.w3.org/2002/ws/arch/

[6] http://www-306.ibm.com/software/solutions/webservices/pdf/SecureReliableTransactedWSAction.pdf

[7] See http://www.ws-i.org/Documents.aspx for background on WS-I and information on WS-I working group charters, including the recently-formed requirements working group.

[8] http://www.omg.org/technology/documents/formal/add_struct.htm

[9] http://www.omg.org/technology/documents/formal/transaction_service.htm

[10] http://www.iona.com/products/appserv-mainframe.htm