

# Transacting Business with Web Services, part 2

## The coming fusion of business transaction management and business process management

■ In the first part of this article (*WSJ*, Vol. 3, issue 9), we examined the need to integrate business transaction management (BTM) software into business process management standards and products. We believe that BTM offers previously inaccessible levels of application coordination and process synchronization, radically simplifying the design and implementation of transactional business processes.

The most promising Web service/XML BPM standardization work is taking place in the OASIS WS BPEL Technical Committee. Members of the committee have been working on proposals to increase and modify the available syntax of the BPEL language to support the use of BTM. A late August submission to the committee ([www.oasis-open.org/committees/download.php/3263/BPEL.and.Business.Transaction.Management.Choreology.Submission.html](http://www.oasis-open.org/committees/download.php/3263/BPEL.and.Business.Transaction.Management.Choreology.Submission.html)) raised issues #53 to #59, which were discussed in detail at the September face-to-face meeting in Redmond, WA. Here we'll look at the state of that discussion, and possible outcomes for the final WS-BPEL standard that will emerge.

We addressed the creation of business transactions in external services and within BPEL processes, and scopes; the propagation of business transaction contexts between services and processes, and the way in which a BPEL process can model a business transaction participant. This article shows the kind of syntax that process designers could use to define business transaction behaviors in their BPEL processes. We briefly



WRITTEN BY  
ALASTAIR GREEN &



PETER FURNISS

show some possible variants that emerged from the recent technical committee discussion. (The new XML elements and attributes referring to business transactions in the code examples below reflect the original submission. They illustrate the principles involved, but may well be altered or rejected by the committee process during the remainder of this year.)

A client, in a pre-existing package or custom application that is external to a BPEL process, can request that the coordination service create a business transaction, receiving in reply a business transaction context. A WS-Coordination `<wscoor:CoordinationContext>` is one example of a

context structure that could be used for this purpose. Equally, a BPEL process should be able to request the creation of a business transaction, storing the resulting context element as the value of a BPEL variable. Appropriate syntax must be added to the BPEL language to support this. (In the example XML that follows, the default namespace is assumed to be that of a future, standardized version of the BPEL language.

Variables to hold business transaction contexts can be declared thus:

```
<variables>
  <variable name=
    "receivedBusinessTransactionContext"
    type="wscoor:CoordinationContext" />
  <variable name=
    "ourBusinessTransactionContext"
    type="wscoor:CoordinationContext" />
</variables>
```

A business transaction context can be created and stored using the following proposed syntax:

```
<businessTransaction action="new"
context="ourBusinessTransactionContext" />
```

If a client invokes a WSDL-defined Web service operation that is offered by a BPEL process, then it should be able to attach a business transaction context to the invocation message, and have that context stored in a variable in the receiving process. This implies additional syntax in the `<receive/>` verb-element:

```
<receive
  partnerLink="customer"
  portType="al:reservationPT"
  operation="createReservation"
  variable="reservationRequest"
  businessTransactionContext=
    "received BusinessTransaction
    Context" />
```

Equally, a BPEL process that holds a context (either by virtue of importing it via a `<receive/>` or by creating it using the new verb-element `<businessTransaction/>`) should be able to transmit that context when it invokes a Web service. This again implies new syntax:

```
<invoke
  partnerLink="airline"
  portType="al:reservationPT"
  operation="createReservation"
  inputVariable="reservationRequest"
  inputBusinessTransactionContext="our
  BusinessTransactionContext"
  outputVariable="reservationDetails"
  outputBusinessTransactionParticipants
  ="FlightComponent" />
```

The purpose of sending a business transaction context to a service is to enable the service to register participants. Registration is carried out using a registration capability that allows coordinator-participant relation-

ships to be constructed (OASIS BTP, WS-Coordination, and WS-CAF all have this feature). Services register participants with the coordination service, not with the invoking application. However, for correlation purposes the identity of the registered participants are returned to the invoker using the attribute `outputBusinessTransactionParticipants`.

The context/participant traffic may also travel in the reverse direction. Sometimes a client makes a request that the service provide a business transaction context, allowing the client to then register as a participant with the service's business transaction. Two new attributes, `<outputBusinessTransactionContext>` and `<inputBusinessTransactionParticipants>` are used to support this advanced behavior.

## Creating and Terminating a Business Transaction

We have already shown you a simple example of creating a business transaction. It should also be possible to create a business transaction that is the child of a parent transaction. This allows transactions trees to be created, which can aid in complex synchronization and ordering cases:

```
<businessTransaction action="new"
  context="ourBusinessTransactionContext"
  parentContext="receivedBusinessTransactionContext">
```

Once a process has finished interacting with the services that carry out a business transaction's work, it must either confirm or cancel the transaction:

```
<businessTransaction action="confirm"
  context="ourBusinessTransactionContext" />
```

This requests that the underlying BTM coordination service confirm all participants. Likewise, a business transaction can be instructed to cancel all of its participants:

```
<businessTransaction action="cancel"
  context="ourBusinessTransactionContext" />
```

A process may wish to terminate a subset of the participants in a transaction. To do this it can use the identity of a participant:

```
<businessTransaction action="confirm"
  context="ourBusinessTransactionContext"
  participants="flightComponent
  hotelComponent" />
```

This instruction implicitly cancels any participants that are not specified.

The BTM coordination service ensures that termination instructions of this kind are correctly and completely delivered, even in the case of temporary process, processor, or network failures. Note that the BTM coordination service may be deployed as part of a BPEL execution engine, or may be freestanding.

## How BPEL Processes Model Business Transaction Participant Behavior

BPEL relates in two ways to participants. A Web service, external to a BPEL process, may act as a participant. And a BPEL process (which presents itself to other processes as a Web service) may also act directly as a participant.

BPEL is not designed to manage persistent

quote, and one to turn that quote into an order (and possibly, one to cancel the quote).

A business transaction participant can use these service operations to model provisional, contingent behavior; and finalization behavior (confirmation, cancellation). In this case, a participant within a BPEL process might invoke `getQuote` on the service as its prepare operation, and would then invoke `cancelQuote` as its cancel operation, or invoke `executeOrder` as its confirm behavior.

The example shows this case, with a BPEL process registering itself as a participant in a business transaction using a received business transaction context (see Listing 1).

New handlers are added to deal with BTM cancel and confirm decisions. The existing fault-handler is triggered if a failure occurs during the forward work – including the case where a BTM cancel instruction is received before the forward work completes. The confirm and cancel handlers are proposed as a supplement to the existing (nontransactional) compensation-handler

“ A business transaction context can be created and stored ”

resources internally within an executable process. BPEL variables are largely intended for control flow and for passing data between the process and ancillary Web services. It is up to vendors to provide implementations of the relevant BTM standards, such as WS-T and BTP, that can be inserted into Web service operations to allow services to correctly interoperate with a coordination service.

There is one important case, however, where a BPEL process is the appropriate place to define participant behavior. Take an environment where an existing application offers operations on its service interface (or where such an interface can easily be added, to allow the application to be accessed as a Web service). It may not be possible to modify or enlarge the application's suite of operations. For example, a CRM or ERP package may offer an operation to create an order, and one to delete an order. Or it may have an operation to offer a

model for local exception processing in BPEL (although a cancel-handler and compensation-handler could be merged).

The registration of the participant is performed by `<businessTransaction action="register">`. This enables triggering of the confirm and cancel handlers by the BTM messages.

All of the new constructs described here would be used both in abstract BPEL processes (which define collaboration protocols), and in executable processes.

## The Changes and Additions Needed for BPEL to Support BTM

In the first part of this article, the authors posed four questions that must be answered in the BPEL standardization process to properly integrate BTM. The answers are being considered in the BPEL community. There is a spectrum of opinion within the committee on how far BPEL should go in recognizing or supporting BTM features in the first version of the standard.

## How Do Business Transaction Coordination Protocols Work?

All transactional coordination protocols have some common features (see Figure 1). Participant systems update business information: each piece of state data must be changed in accordance with the instructions of a central coordination service. The coordination service is in turn the servant of an assembly application, and of a terminating application. (Frequently the assembler and the terminator are fused in a single controlling application.)

The assembler requests that the coordination service create a coordination or transaction, and tags its communications with participants with a transaction identity and the address of the coordination service (this is termed propagation or infection).

The participants then signal the coordination service that they are prepared to be instructed by the coordination service. Typically this means that they have effected provisional, reversible state changes. (Participants may also communicate that they have failed to prepare, either for business or technical reasons. In this case they are not available to the terminator for inclusion in the final outcome.)

The terminator now decides what completion instructions should be communicated to each participant. Typical examples of mutually exclusive completion instructions are confirm and cancel. (These are the conceptual instructions used by BTP and WS-T; future, special-purpose

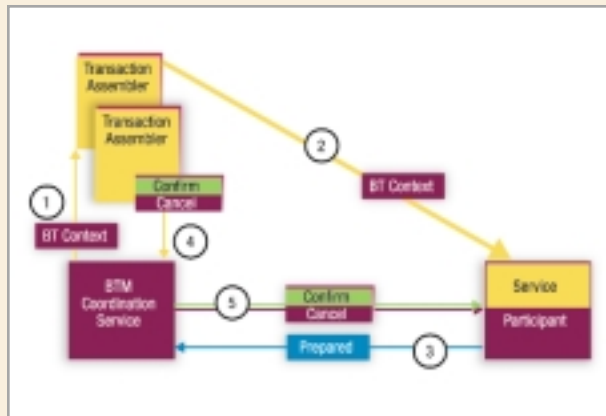


FIGURE 1 | Business transaction coordination protocols at work

coordination protocols might employ a larger set of possible outcomes.)

Another way of looking at this progression is shown in Figure 2. A BTM participant moves from an active state to a prepared state (where it has carried out provisional work), and then to a final state (either confirmed or cancelled). The

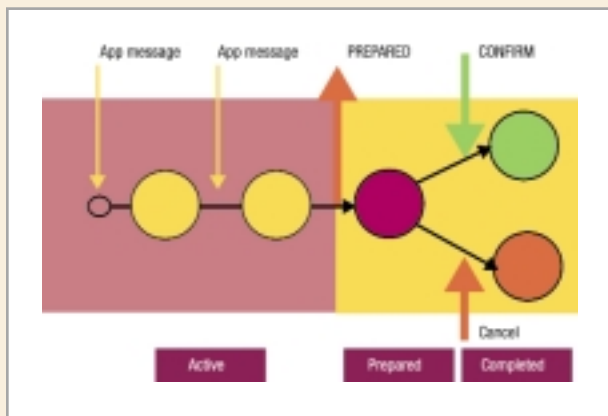


FIGURE 2 | State transition diagram

state transition diagram shows how application messages and BTM protocol messages cause these state changes. When a participant receives its completion instruction it takes whatever internal action is needed to conform with the semantics of the instruction for the particular type of business transaction being performed.

When a participant receives its completion instruction it takes whatever internal action is needed to conform with the semantics of the instruction for the particular type of business transaction being performed.

When a participant receives its completion instruction it takes whatever internal action is needed to conform with the semantics of the instruction for the particular type of business transaction being performed.

- **How do you propagate business transactions between Web services and business processes?**

A business transaction context needs to travel between the parties for propagation. The committee is divided on how this behavior should be described or specified by process designers. An “implicit” approach would involve either runtime configuration or design-time marking of Web service invocations as being “transactional.” The “explicit” approach makes context transmission visible at the BPEL level. (The implicit approach using deployment-time configuration may be very difficult to understand, and probably presents significant problems in defining abstract processes.)

Either way, the parties need to agree on the type of business transaction context being communicated. BPEL should permit this choice at deployment time, not design time. The representation of the agreed context “on the wire” is achieved by WSDL’s ability to map an abstract message part to a type and position within a concrete message.

- **How do you initiate and terminate new business transactions within a business process?**

Provide new verb-elements in BPEL: <businessTransaction>. Some technical committee members have suggested that business transaction creation and termination should occur implicitly when scopes are entered and left. This would require some construct to mark scopes as “transactional.” This is tied to the notion of implicit context transmission.

- **How do you propagate business transactions between business processes and nested scopes?**

If an explicit approach to creation/termination is preferred, then simply allow nested scopes to use a context variable that has been declared and assigned in an outer scope/process. If an implicit approach is adopted, then inner scopes would inherit the business transactional context of their outer scope.

- **How do you define the reaction of processes and subprocesses to the progress of a business transaction?**

Allow a process to register itself as a participant, using the <businessTransaction action=“register”> variant of the new verb-element. There seems to be little appetite for permitting scopes within processes to act as BTM participants.

ness transaction the system would change the reservation status to confirmed, and trigger related fulfillment and billing/payments processing. Note that the participating service fully controls its own internal implementation of the prepare, confirm, and cancel operations; these are written to produce an externally visible effect that is compatible with the overarching business contract that the transaction is effecting. The assembler/terminator applications are unaware of the participant's implementation; the whole interaction is conformant, in BPEL terms, with an abstract process that describes the contents and legitimate sequences of process-to-process messages.

The terminating application uses its business rules to orchestrate the completion of the business transaction. Such rules will determine whether there is a viable set of participants, or whether a critical participant's inability to prepare has vitiated the whole transaction. The terminator's rules may be used to select a subset of possible participants for confirmation, discarding the unwanted remainder, or it may simply confirm all of the participants. (This ability to manipulate the final population of participating services is known as a "cohesive business transaction" or "cohesion" in BTP terminology. WS-T BA can be used to implement similar behavior.)

## “ BPEL variables are largely intended for flow control and for passing data between the process and ancillary Web services ”

In addition, we expect that considering the standardization of BPEL's interaction with BTM protocols will increase the desire to achieve a single, agreed BTM standard (perhaps to be called WS-Business Transaction?). The current confusion between BTP (an OASIS Committee Specification) and WS-Coordination plus WS-Transaction (draft proprietary specifications from three key vendors) needs to be cleared up so that end users can stop worrying about the current standards flux and implementers can get products to market more quickly. (The recent emergence of yet more draft proprietary transaction management specifications in the WS-Composite Application Framework accentuates the need.) We believe that there should be wide agreement that the fundamental two-phase outcome principles of BTP and WS-T Business Activity align sufficiently to end up with a common, single standard for Web service transactions. (In our view, WS-T Atomic Transaction duplicates, as a special case, the capabilities of WS-T BA, and is therefore equivalent.)

Given clarity on the standard for Web service

business transactions, and the proposed BPEL revisions, end users will be much better able to realize the promises of BPM (relative simplicity and reliability), in environments that support the processing of valuable economic transactions. ©

### ■ About the Authors

Alastair Green is CEO/CTO and cofounder of Choreology Ltd, the business transaction management (BTM) company. A coauthor of the OASIS Business Transaction Protocol standard and an active member of the OASIS WSBPEL committee, he has spent over 20 years helping to produce distributed transaction products for software vendors, and deploying them in end-user business processes, particularly in banks.

■■■ [alastair.green@choreology.com](mailto:alastair.green@choreology.com)

Dr Peter Furniss is chief scientist at Choreology Ltd, where he is responsible, along with Alastair Green and Tony Fletcher, for Choreology's involvement in OASIS BTP, and is the editor of the BTP specification. Peter is a cofounder of Choreology and was formerly at HP Arjuna Labs, working as a product architect on the failure-recovery aspects of Arjuna's Java transaction service

■■■ [peter.furniss@choreology.com](mailto:peter.furniss@choreology.com)

### Listing 1

```
<process>
  <variables>
    <variable name="receivedContext" ... />
    <variable name="quoteParticipant" ... />
  </variables>
  . . .
  <!-- handlers appear in the script before the forward
work -->
  <faultHandler/> <!-- clean up partial provisional
work -->
    <!-- no op: getQuote operation failure
leaves no trace -->
  <cancelHandler> <!-- cancel prepared, provisional
work -->
    <invoke operation="cancelQuote" . . . />
  </cancelHandler>
  <confirmHandler> <!-- finalize prepared, provisional
```

```
work -->
    <invoke operation="executeOrder" . . . />
  </confirmHandler>
  <!-- the forward work>
  <receive . . .
    businessTransactionContext="receivedContext">
    . . .
  </receive>
  <businessTransaction action="register"
    registerWith="receivedContext"
    registeredAs="quoteParticipant">
  <!-- provisional work, implicit prepare -->
  <invoke
    operation="getQuote" . . . />
  <reply
    businessTransactionParticipant="quoteParticipant"
    . . . >
  </process>
```

Download the code at  
[sys-con.com/webservices](http://sys-con.com/webservices)