

BPEL and Business Transaction Management: Choreology Submission to OASIS WS-BPEL Technical Committee.

Authors (alphabetical order)

[Tony Fletcher](#), Choreology Ltd
[Peter Furniss](#), Choreology Ltd
[Alastair Green](#), Choreology Ltd
[Robert Haugen](#), Choreology Ltd

Copyright © Choreology Ltd, 2003. This submission is made subject to OASIS IPR Policy.

The authors are not aware of any proprietary interest by them, severally or jointly, or by Choreology Ltd, in the contents of this submission, other than Choreology Ltd's copyright.

Introduction

An overall motivation for this submission is given in an article by one of the authors, Alastair Green, in the September issue of [Web Services Journal](#).

We begin with an “Ur-issue”:

(This issue and the others distinguished in this proposal have been entered in the [WSBPEL TC issues list](#). The issue descriptions, other than the Ur-issue are deliberately very brief, and the text after each description gives further details and outlines proposed syntax changes).

BTM Issue A ([BPEL issue 53](#)) : Desirable for WS-BPEL to include Business Transaction Management (BTM) programming constructs which are compatible with WS-T, BTP and WS-TXM.

Description

There are three multi-vendor specifications which address the needs of business transaction management for Web Services: Business Transaction Protocol 1.0 (OASIS Committee Specification, June 2002); WS-Transaction (proprietary consortium, August 2002), and the very recently published WS-TXM (proprietary consortium, August 2003).

In our view BTP Cohesions, WS-T Business Activity, and WS-TXM Long-Running Actions are the most relevant aspects of these specifications for WS-BPEL. These aspects overlap to a very high degree, each effectively utilizing a two-phase (promise/decide) outcome protocol. (We should emphasize that there has been little time to analyze or assimilate WS-TXM, so this is a provisional conclusion with respect to that specification).

WS-BPEL should be equipped with the ability to create and terminate business transactions, and to define process participation in such transactions, in a way which is compatible with the intersection of these three capabilities. This will minimize dependence on future standardization efforts in the BTM area.

Submitter's proposal

Syntax should be added to WS-BPEL to permit

- 1. Business transaction creation, manifested by the creation of a business transaction context variable.*
- 2. Business transaction context reception and transmission (propagation) via invoke/receive/reply, allowing sub-transaction or pass-through behaviour.*
- 3. Process involvement in business transactions as participants, which can handle positive and negative transaction finalization messages (by means of confirm and cancel handlers).*
- 4. Two-phase termination of business transactions by selection of participants to be prepared, to be cancelled and to be confirmed.*

These facilities are all portable over products implementing WS-T BA, BTP and WS-TXM outcome protocols.

The WS-Coordination specification (proprietary consortium, August 2002) is able to support context passing for multiple coordination protocols, and its use will support the goal of compatibility with WS-T, BTP and WS-TXM.

Business-transaction aware web-services

It should be noted that a “business transaction” is normally performed in support of some economic transaction – that it coordinates actions that have an effect on the parties and their relationships that go beyond the lifetime of the transaction itself. Since a BPEL process cannot directly manipulate data with a lifetime longer than the process, but always delegates to a web-service, the invoked web-services will either themselves be participants in the business transaction (strictly, the invocation will trigger the registration of participants) or the BPEL process will register as a participant and then make non-transaction invocations on other web-services. In the former case, the invoked web-services are “business-transaction aware”; the BPEL process will export the context to it and the web-services will implement the transactional responsibilities internally.

Similarly, a BPEL process, as an offerer of a web-service, may import a context from a non-BPEL application – in which case it is itself a business-transaction aware web-service from the perspective of its caller – and either registers as a participant or passes the context on in its own invocations.

Of course, since a web-service interface does not distinguish externally how it is implemented internally, the concerns of BPEL's business transaction support can effectively be confined to the single process – contexts can be created, used, imported and exported but where the imported ones come from and where the exported go to is not a concern of the language (though it may be a concern of the BPEL programmer).

Proposed BPEL syntax for Business Transaction Management

BTM Issue B ([BPEL issue 54](#)) : *Construct to hold business transaction contexts.*

Description: A mechanism is needed to hold business transaction contexts and participant identifications.

The main construct for handling a Business Transaction/Business Activity within BPEL is a business transaction context (btContext). This is a value that represents a business transaction and is held in a variable, appropriately typed in the `variables` element:

```
<variables>
  ...
  <variable name="receivedContext"
type="wscoor:CoordinationContext" />
  <variable name="flightTransactionCtx"
type="wscoor:CoordinationContext" />
  <variable name="trainTransactionCtx"
type="wscoor:CoordinationContext" />
</variables>
```

Similarly, a BPEL construct `businessTransactionParticipantIdentification` (btParticipantId) is defined. This is a value that identifies a participant within the scope of the business transaction it is registered with. Like a btContext, a btParticipantId value is held in an appropriately typed BPEL variable.

Both btContext values and btParticipantId values need to be associated with application (web-service) messages sent and received by a BPEL process, or, in some cases, with just parts of sent or received messages. The constructs to indicate this – i.e. to specify which BPEL variables are used to set a value on a sent message and which variables are set from a received message – take advantage of the query attribute mechanisms of propertyAliases. As with correlation sets, it is assumed that message properties and propertyAliases are defined that map from the particular fields and headers of a message

to the business transaction constructs. ([Appendix C – contexts and bindings](#) discusses some issues concerning how the presence of a context is represented in WSDL).

(Note – in the BPEL 1.1 specification, properties can only be simple XML schema types, whereas a business transaction context will be a complex type (typically containing at least an unambiguous identifier and registration endpoint reference, and perhaps qualifying information about sub-protocol, timeouts etc.). [WS-BPEL issue 12](#) proposes enhancing the property mechanism to allow non-simple types for properties.

Mapping of the btContexts and bTParticipantId between the BPEL variables and the sent or received messages is defined in attributes of the receive, invoke, onMessage and reply elements.

It is quite possible to have multiple btContexts handled within a single BPEL process – typical cases are when a subordinate business transaction is initiated (as nested or explicitly interposed) and when multiple transactions are manipulated as alternatives. A particularly useful case is when both of these occur – multiple child transactions are created to deliver the application requirement of a received transactional request, only one child eventually being confirmed (and then only if the received transaction is confirmed).

BTM Issue C ([BPEL issue 55](#)) : *business transaction propagation*

Description: Constructs are needed to specify how business transaction contexts are transmitted and received with the application messages sent and received via invoke, receive, reply etc.

Receive business transaction information on an inbound message

This construct is needed to define where the received btContext or bTParticipantId is held, such that it can later be used to pass on outbound messages used for registration.

```
<receive partnerLink="customer"
  portType="SP:purchasingPT"
  operation="purchase" variable="PO"
  businessTransactionContext="receivedContext"
  >
  ...
</receive>

<receive partnerLink="supplier_one"
  portType="SP:quotationPT"
  operation="requestForQuote" variable="quote"
  businessTransactionParticipant="participant_at_one"
  >
  ...
</receive>
```

The identical constructs can be present in onMessage.

Pass business transaction information on an invoke

For a synchronous invoke, there can be business transaction information on either or both of the request and the response, so the business transaction values are distinguished by which way they are going (from the perspective of the invoked service)

```
<invoke partnerLink="airline"
  portType="AL:bookingPT"
  operation="book"
  inputVariable="flightRequest"
  inputBusinessTransactionContext="flightTransaction"
  >
  ...
</invoke>
```

A very common case would be to supply btContext on a request, and the invoked service creates and registers a participant, and returns the ParticipantIdentification on the response.

```
<invoke partnerLink="hotel_sussex"
  portType="HTL:reservationPT"
  operation="book"
  inputVariable="hotelRequest"
  inputBusinessTransactionContext="flightTransaction"
  outputVariable="hotelResponse"
  outputBusinessTransactionParticipant="hotelAtGatwick"
  >
  ...
</invoke>
```

Other possible cases include a request with no business transaction information, but the invoked service initiates a BT and returns the btContext on the response. (There is no general requirement that the control of a business transaction should be in the same direction as the initiative for the application work – although it is very common to have a client (or client-side) application initiate the work and control the transaction, it is quite possible for an application to initiate work in which creates a transaction in which the application itself (or part of it) registers as a participant):

```
<invoke partnerLink="secretariat"
  portType="TA:meetingArrangementPT"
  operation="meetingRequest"
  inputVariable="meetingProposal"
  outputVariable="meetingDetails"
  outputBusinessTransactionContext="meetingTransaction"
  >
  ...
</invoke>
```

Pass a context or participant identification on a reply

Similar to invoke, but without the need for an input/output distinction.

```
<reply partnerLink="customer"
  portType="HTL:reservationPT"
  operation="book"
  variable="bookingDetails"
  businessTransactionParticipant="ourParticipantId"
  >
  ...
</reply>
```

BTM Issue D ([BPEL issue 56](#)) : *Business transaction creation*

Description: Constructs are needed to specify how a business transaction is initiated. This will cause the creation of a new, propagatable context value.

Begin a Business Transaction/Business Activity

This is used to create a new business transaction, loading a context into a variable, which can then be passed on invoked web-services.

The <businessTransaction> element is an activity in its own right. It would often appear in a <sequence>.

There are various properties that could be defined for the BT – exactly what these are may depend on the capabilities of the underlying implementation and BT protocols.

To create a new business transaction, and put the resulting btContext in the variable “trainTransaction”:

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

A new transaction can be created that is itself subordinate to an existing one:

```
<businessTransaction action="new"
  context="taxiToAirportSubtransaction"
  parentContext="flightTransaction" />
```

BTM Issue E ([BPEL issue 57](#)) : *business transaction termination*

Description: Constructs are needed to specify how a BPEL process requests the termination (completion) of a business transaction. The process needs to be able to specify that completion is negative (cancelled/compensated/undone) or positive (confirmed).

BTM Issue F ([BPEL issue 58](#)) : *Selective termination of business transaction participants*

Description: BTM termination constructs should allow the selection of registered participants, such that some can be confirmed, some cancelled

Finish a Business Transaction/Business Activity

There are various actions possible. The simplest is the obvious “final” confirm or cancel for the whole business transaction. An instruction to confirm a business transaction is only a request that the participants are all confirmed – the actual result of the transaction depends on the protocol exchanges with the participants:

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

The flexible nature of business activities/cohesions make it appropriate to select subsets of the registered participants to be cancelled or confirmed. In this example, two of the participants (identified using the participantIdentification values received for them) are cancelled, and then the rest are confirmed:

```
<businessTransaction action="cancel"
  context="flightTransaction"
  participants="trainToGatwick hotelAtGatwick">

<businessTransaction action="confirm"
  context="flightTransaction">
```

It would also be possible to select only the Participants to be confirmed.

```
<businessTransaction action="confirm"
  context="flightTransaction"
  participants="taxiToHeathrow flightFromHeathrow">
```

Explicitly asking for some participants to receive a prepare signal is also possible:

```
<businessTransaction action="prepare"
  context="trainTransaction"
  participants="taxiInGlasgow">
```

Since a business transaction can confirm some of its participants and require or accept the cancellation of others, it is necessary to have a way of determining the current status of a participant:

```
<businessTransaction action="query"
  status="glasgowTaxiStatus"
  context="trainTransaction"
  participants="taxiInGlasgow">
```

[tbd – there need to be defined faults for the alternative (unexpected/undesired) outcomes for these actions]

Means to query the status of a Business Transaction/Business Activity

Just

```
<businessTransaction action="query" outcome="answerVariable">
```

BTM Issue G ([BPEL issue 59](#)) : *BPEL process as business transaction participant*

Description: Constructs are needed to allow a BPEL process to register as a participant in a business transaction, such that the work of the process is confirmed or cancelled in reaction to the appropriate business transaction protocol messages.

Participant definition - how a process responds to transaction events

In many cases, the participants in a business transaction will all be web-services implemented in something other than BPEL, but supporting the protocol. However, a BPEL process can itself be defined as a Participant. The normal, forward action of the process will establish a provisional state, and the transaction completion exchange will cause the process to be finalised by triggering the appropriate handler. A process is declared as a Participant by including an element that states which (received) context it will use for registration and defining handlers for the various transaction events. For example:

```
<process>
  <variables>
    <variable name="theContext" ... />
    <variable name="myIdentity" ... />
  </variables>

  <faultHandler> .... </faultHandler>
  <confirmHandler> .... </confirmHandler>
  <cancelHandler> .... </cancelHandler>

  <receive . . .
    businessTransactionContext="theContext"
  >
  ...
</receive>

  <businessTransaction action="register"
    registerWith="theContext"
    registeredAs="myIdentity"
  >

  ... other processing as needed ...

  <reply . . .
    businessTransactionParticipant="myIdentity">
  ...
</reply>
```

</process>

The <businessTransaction> activity with action="register" causes a participant to be registered in the business transaction identified by the context in the registerWith variable. If the registeredAs attribute is present, the btParticipantIdentity value is captured in the registeredAs variable. This value can then be returned on an outbound message via the businessTransactionParticipantIdentity attribute.

The act of registering the process as a participant means that, if the normal activity of the process completes (i.e. none of the faultHandlers fire), then either the confirmHandler OR the cancelHandler will be invoked, via the business transaction protocol.

(See [Appendix B – is it always compensation](#) for further discussion of the names of the handlers)

Only one <businessTransaction action="register"> activity can be executed in the lifetime of a business process instance (the fault bpws:repeatedRegistration is thrown)

A common pattern for a process, which receives a context and registers the process as a whole as a participant is shown in [Example 1 – BPEL process initiates business transaction, all participants are in invoked web-services](#) in [Appendix A - examples of suggested BPEL syntax for supporting business transactions](#).

The processing of a fault-handler in a process that has been registered as a participant will cause the bt protocol failure (cancelled) message to be sent to the business transaction. (Exactly what this message is depends on details of the protocol – the semantic is that the work of the participant has NOT been performed and the participant is not now expecting to be involved in the transaction)

If the business transaction (coordinator) instructs the registered participant to cancel before the process completes normally (i.e a cancellation or abort instruction is received between registration and the normal forwards completion of the process), a fault bpws:forcedCancellation occurs, and will be caught by the appropriate (or default) faultHandler.

If a cancellation instruction is received after the normal completion (which is synonymous with the participant reaching it's "prepared"/ "ready"/ "provisionally completed" state), the cancelHandler will be invoked.

Note - whether the same or different messages are used for cancellation before and after the prepared point is a point of detail of the transaction protocol. However, the concern here is with the handling in the BPEL process of a delivered transactional semantic, which can be regarded as invariant over the differences between the protocols.

Acknowledgement

The authors thanks Bill Pope for his advice and comments in the preparation of this proposal.

Appendix A - examples of suggested BPEL syntax for supporting business transactions

These examples are partial – the details of where the business transaction information is carried and how it is copied, the correlation sets and various other things essential to a complete BPEL document are not included.

Example 1 – BPEL process initiates business transaction, all participants are in invoked web-services

A BPEL process accepts a request for a combination of two related items – “nuts” and “bolts”, then initiates a business transaction and invokes requests on two webservices, one for each item, propagating the btContext on both requests. The invoked webservices offer the same interface, but the particular values will be different. It then requests confirmation of the business transaction. If the business transaction fails (cancels), the received request is rejected.

```
<process name="nutandbolt">
  <partnerLinks>
    <partnerLink name="customer" ..../>
    <partnerLink name="nutSupplier" ..../>
    <partnerLink name="boltSupplier" ..../>
  </partnerLinks>

  <variables>
    <variable name="combinationOrderRequest"
messageType="hdcmb:orderRequestMsg" />
    <variable name="combinationOrderAccepted"
messageType="hdcmb:AcceptanceMsg" />
    <variable name="combinationOrderRejected"
messageType="hdcmb:RejectionMsg" />
    <variable name="nutOrderRequest"    messageType="hdwr:requestMsg"
/>
    <variable name="nutOrderReply"
messageType="hdwr:responseMsg" />
    <variable name="boltOrderRequest"    messageType="hdwr:requestMsg"
/>
    <variable name="boltOrderReply"
messageType="hdwr:responseMsg" />
    <variable name="ourBizTran" type="wscoor:CoordinationContext" />
  </variables>

  <!-- if anything goes wrong, including the business
transaction cancelling, we must reply to the
the invoker -->

  <faultHandler>
    <reply partnerLink="customer"
```

```

        portType="hdcmb:consistentCombinationPT"
        operation="orderRequest"
        variable="combinationOrderRejected" />
</faultHandler>

<sequence>
  <receive partnerLink="customer"
    portType="hdcmb:consistentCombinationPT"
    operation="orderRequest"
    variable="combinationOrderRequest" />

  <businessTransaction action="new"
    context="ourBizTran"/>

  <!-- copy values from the received message to the
    sent messages and the reject message -->

  <!-- following could be done in parallel using flow -->

  <invoke partnerLink="nutSupplier"
    portType="hdwr:orderingWithTxPT"
    operation="order"
    inputVariable="nutOrderRequest"
    inputBusinessTransactionContext="ourBizTran"
    outputVariable="nutOrderReply"
  >

</invoke>

  <invoke partnerLink="boltSupplier"
    portType="hdwr:orderingWithTxPT"
    operation="order"
    inputVariable="boltOrderRequest"
    inputBusinessTransactionContext="ourBizTran"
    outputVariable="boltOrderReply">
  >
</invoke>

  <businessTransaction action="confirm"
    context="ourBizTran"/>

  <!-- copy values from the replies
    to our reply to the customer -->

  <reply partnerLink="customer"
    portType="hdcmb:consistentCombinationPT"
    operation="orderRequest"
    variable="combinationOrderAccepted" />

</sequence>
</process>

```

That example shows a simple business transaction, used in an atomic manner – there are only two participants and both are required for success.

Example 2 – BPEL receives a btContext, creates and registers a Participant

This example might be used to make a business-transaction-capable “cover” over a web-service that does not support business transaction protocols, but offers operations that allow it to behave appropriately. The “existing” service has operations to

- make a tentative reservation
- confirm a tentative reservation
- cancel a tentative reservation.

Tentative reservations automatically cancel if not confirmed after some time (the explicit cancellation is a courtesy). The new, BPEL-defined service has exactly the same signature for the “makeReservation” request as the underlying one, except that it requires a btContext in the headers of the incoming request and returns the identification of the participant in the headers of the response. The confirm and cancel messages to the existing service use a reservation reference that was returned in the reply to the request. This reference is purely informational for the user of the new transactional service offered by the BPEL process.

```
<process name="transactionalReservation" ... >

  <partnerLinks>
    <partnerLink name="existingService" ... />
    <partnerLink name="user" ... />
  </partnerLinks>

  <variables>
    <variable name="reservationRequest"
messageType="htl:reservationRequestMsg" />
    <variable name="reservationConfirmation"
messageType="htl:reservationConfirmMsg" />
    <variable name="reservationCancellation"
messageType="htl:reservationCancelMsg" />
    <variable name="transactionContext"
type="wscoor:CoordinationContext" />
    <variable name="ourParticipant"
type="wscoor:ParticipantProtocolService" />
    <variable name="reservationDetails" type=xsd:String />
  </variables>

  ...

  <faultHandler />

  <confirmHandler>
    <!-- copy reservation identification from
         reservationDetails to reservationConfirmation -->
    <invoke partnerLink="existingService" ...
      operation="confirm"
      inputVariable="reservationConfirmation"
    />
  </confirmHandler>
</process>
```

```

</confirmHandler>

<cancelHandler>
  <!-- copy reservation identification from
       reservationDetails to reservationCancellation -->
  <invoke partnerLink="existingService" ...
         operation="cancel"
         inputVariable="reservationCancellation"
         />
</cancelHandler>

<sequence>
  <receive partnerLink="user"
          portType="ourSvc:transReservationPT"
          operation="makeReservation"
          variable="reservationRequest"
          businessTransactionContext="transactionContext">
  </receive>

  <businessTransaction action="register"
                      registerWithContext="transactionContext"
                      registeredAs="ourParticipant">

  <invoke partnerLink="existingService"
          operation="makeReservation"
          inputVariable="reservationRequest"
          outputVariable="reservationDetails"
          />

  <reply partnerLink="user"
         portType="ourSvc:transReservationPT"
         operation="makeReservation"
         variable="reservationDetails"
         businessTransactionParticipantIdentification="
ourParticipant" >
  </reply>

</sequence>

</process>

```

If the underlying service does not automatically cancel tentative reservations, a failure in this process or the receipt of a cancellation instruction via the business transaction protocol after invoking makeReservation could cause a tentative reservation that is orphaned. The faultHandler can be expanded to perform a cancellation – however, since the reservation identification is not yet available, it would have to identify the reservation by some of the user-supplied fields.

Appendix B – is it always compensation ?

The proposal uses the name cancelHandler, rather than compensationHandler for the “negative completion” handler. This reflects a view that between separately defined participants involved in a business transaction, compensation is just one approach a

participant may use to fulfil its promise to obey the transaction decision. (For a completed inner scope, different considerations apply, since the relationship between the inner and outer scopes is much closer than between separate processes.)

It is widely asserted that “classic” two-phase commit is inappropriate for loosely-coupled, web-service based interactions. Even where all the systems involved are under common ownership, the design assumptions typical of most transaction systems and databases do not fit – transactions may take some considerable time and lock-holding would embarrass the systems. This becomes even more the case where the systems are under different ownership, as in true business-to-business interactions.

However, a multi-step business process, involving interactions with multiple web-services still needs to have some level of coordination – if some parts of the process do not succeed, other parts, which might themselves be able to succeed need to be undone, not done or reversed. Business transaction protocols are of course concerned with communicating the signals to enable this. A common assumption is to say that the only solution is to let the transactions involved in each part run to completion, and then, if the work has to be cancelled, run a compensating counter-operation.

However, when applied to a business transaction protocol, this approach makes assumptions that are inappropriate to the loosely-coupled case. One of these is that participants achieve some intermediate state of “completion” (as signalled by WS-T BA “Completed”), which is identical to the final (un-cancelled) state in application terms, although distinguished from it at the protocol level. The protocol distinction is that, the protocols in question (WS-T BA, BTP, WS-TXM) all have an additional signal (“close” in the case of WS-T BA), which moves the protocol state to a “truly final” state. *Ipsa facto*, there are distinct “provisional” and “truly final” states, at least as known to the protocol. Allowing an implementation of a “participant” (i.e. the application service or responder side) to have distinct application states corresponding to these gives flexibility and advantage, but need not, in general, be made visible to the coordinator (or client side). The only understanding that must be shared is that the “contract” implicit in the communication of the application messages within the context of the business transaction ends up either finalised – beyond reversal as far as this business transaction is concerned - or the contract is not made. There may be various levels of qualification as to exactly what finalisation or cancellation entails – but these are themselves part of the “terms and conditions” of the contract.

In this light, the proposed BPEL support for Business Transactions assumes that a BPEL participant should have both a “confirmHandler” and a “cancelHandler”, precisely one of which will be triggered by the completion exchanges of the business transaction protocol. The nature and content of these is entirely an issue for the writer and owner of the scope they are part of. If, in a particular case, a “pure” compensation approach is used – i.e. the provisional state and the truly final state are identical from the application perspective - the “confirmHandler” will be empty. But this is just one option among several.

It should be emphasized that the confirm/cancel approach is a question only for the participant – it is precisely not an aspect of an “exotic transaction model”. Indeed, the concept that all parties in a business transaction should be aware of the internal choices and mechanisms of each other would seem to be contrary to the loosely-coupled approach that is a major motivation for web-services. From that viewpoint, whether a participant uses a “do-compensate” or “validate-do” or “provisional-do” mechanism is a private concern. To be sure, in some cases, the systems involved are all under common design and management, and the internals of each are known – but the use of a generalised protocol which allows each system to meet its internal and external requirements is just as appropriate to such cases as to the more loosely-coupled cases where only the “contract” is in common.

Appendix C – contexts and bindings

The question of how to state that a web-service expects or requires particular headers has been raised in several standards committees. In general, there would seem to be

(at least) two quite distinct ways of saying in WSDL, “I expect to see XYZ headers”:

- a) In a regular binding, which defines the mapping of WSDL parts of the abstract message to the header – the context is then represented as a WSDL part in the port-type definition, and the particular binding specifies that context goes in the header
- b) In a special binding – there is an named binding that references a (text) specification that defines (or references) the syntactic and semantic requirements

Approach a) is very flexible and does not require any special handling by the web-service tools, but exposes the details of the enhanced service to the user. Approach b) may be further extended (as in WS-Policy)

There are some further questions of how much of the semantics of the header is defined by either of these – does the presence of a WS-Coordination context (for example) mean that the web-service **MUST** register a participant, or only that if it determines that it wants to register a participant, this is the context to use; is the service at liberty to register more than one participant ? (Since the web-service architecture does not mandate particular internal behaviour, it is quite difficult to state the implied shared semantics of the context as a protocol element in an unambiguous manner).

However, for BPEL the semantic question is simpler – since the BPEL program defines what the behaviour is there are no architectural barriers to specifying exactly what effects the presence of the context has (although, if these are just to pass the context on in further web-service invocations, the question is merely postponed). [The situation with a BPEL abstract process may be worth considering further here – an abstract definition might (or might not) be a useful way of stating the shared semantics of passing a context]

However, it will be necessary for BPEL to express the extraction and propagation of a context whichever approach is used by the WSDL, since at least the WSDL of invoked services may be a “given” to the BPEL programmer. In addition, the BPEL constructs need to cope with “explicit propagation”, where the business transaction value is inside the application message (particularly likely with the multiple participant identification case – (e.g. a request for quote for a list of items might receive multiple quotes from a single invocation with different combinations of items, each quote handled by a separately registered participant)).