

# Transaction Support in Web Services

**IS660G Web Services**  
**Fall 2004**  
**Prof. Kevin Burns**

**Team 2**  
Ian Loe  
Pratima Vijayakumar  
Simon Chan  
Zheng Qin

---

## Introduction

To the general public, the term “transaction” is used to mean a business exchange where money is traded for goods. But in the domain of computing, the meaning of “transaction” is more specific. Transactions are a fundamental abstraction in dependable computing systems. Its inherent value is to allow programmers to make something that went wrong appear as if it never happened. A transaction implies that a group of activities is completed as a unit, so they all succeed or all fail together. In other words, it’s a semantic that means “all or nothing”.

While transaction processing has been around for many years, spanning many transaction processing systems from CICS to Tuxedo and Microsoft DTC, the underlying technologies all have one pervasive notion: ACID. ACID is an acronym that represents the terms Atomic, Consistent, Isolated, and Durable. (See Table 1)

## ACID Transactions

### Atomic

The transaction will succeed completely or fail completely. This is particularly important when executing business logic that involves the updating of multiple underlying data sources, where the atomicity property turns a set of operations into a single indivisible logical operation.

### Consistent

The data store changes over time through a sequence of consistent states. That is, any data that has been updated during the lifetime of the transaction is left in a consistent state at the end of that transaction, irrespective of any failures that have occurred during the transaction.

### Isolated

The effects of a transaction should be invisible to and isolated from other transaction until it has been committed. This also means that any running transaction believes that it has exclusive access to the resources associated with it.

### Durable

The results are guaranteed to be stored after the transaction completes. That is, once a transaction has been completed, the resulting changes must not be lost even if hardware or software fails.

*Table 1 : Acid Transactions*

## **ACID and Compensating Transactions**

ACID transactions are critical to business interactions. Why? For example, when you deposit some money into your bank account or withdraw some money from it, the bank wants to ensure that your account is credited exactly once for a deposit, and you want to ensure that your account is debited exactly once for a withdrawal. Likewise, when you make changes in the data stored in your application, it must be accessed only when it is internally consistent - transactions guarantee that consistent view.

To ensure consistency, typically all database entries being used by an ACID transaction are locked for the duration. If a transaction fails, the database state is rolled back to its previous state. This capability is provided by database vendors.

While locking will work within a closed system like a database management system, it is not feasible to have it work across enterprises. Take the following scenerio as an example: when you make a hotel reservation, your travel agent cannot lock the hotel's reservation database for as long as the reservation exists (or even as long as a phone call) or the system would grind to a halt. Beyond the technical reasons, it is also improbable that you would be able to control access in a system outside your own enterprise. A more probable scenerio would be an ACID transaction local to each party, i.e. there would be an ACID transaction local to the hotel chain's database that would perform the tasks of updating the room inventory, add the information to the reservation tables and generate a confirmation number, all as a single unit of work. And if the travel agent needs to undo that reservation, a compensating action is taken.

Compensation is specific to the way business data is managed, so it's always part of business logic. This is very different from the automatic rollback provided by databases for ACID transactions. Compensation avoids another problem. Locking of your company's data by anyone on the Internet allows denial-of-service attacks. Using compensation means that your data isn't locked for a long time, but we can no longer have ACID transactions - at least the Isolation guarantees must be relaxed - because the data is visible between the initial change and the compensation.

In effect, one trades softening of the ACID guarantees for flexibility, safety, and control over one's own data. While ACID transactions can be achieved in closed systems, the nature of Web Services applies some limit as to how far you can apply the principles of ACID. The objective of our project is to examine how transactions can be support in Web Services and what is the impact on ACID properties.

## **Standards in the Brew**

There are currently a few standards working towards some form of reliable transaction management using Web Services. The better known ones are the OASIS Business Transaction Protocol, push by Arjuna Technologies Ltd. (formally part of Hewlett-

Packard Company), Fujitsu Limited, IONA Technologies Ltd., Oracle Corporation, and Sun Microsystems, Inc.; and WS-Transaction push by IBM, Microsoft and BEA,

## **OASIS Business Transaction Protocol (BTP)**

### **Background**

The OASIS Business Transaction Technical Committee (BTTC) published the BTP 1.0 specification for coordinating transactions between applications controlled by multiple autonomous parties in May 2002. This is the result of the work of several companies (Sun, HP, Choreology, ORACLE, and others) that hopes to become a standardized Internet-based means of managing complex, ongoing business-to-business (B2B) transactions among multiple organizations. The full BTP specification can be found at <http://www.oasis-open.org/committees/business-transactions>.

One of the key design factors of BTP is that in a business transaction, no single party controls all resources needed. In such an environment, it is assumed that the respective parties manage their own resources but the activities will be coordinate in a defined manner to accomplish the work scoped by a transaction. There is also a provision for individual service providers to decide if they want to be part of a transaction or not; if they decide that they want to be part of the transaction, they must provide a mechanism to confirm or cancel their commitments to the transaction. BTP also allows service providers to autonomously decide when to unlock resources they hold and/or whether to use compensating transactions to roll back transient states that were persisted.

Although not design specifically for Web Services, the protocol will be especially useful in a Web Services environment. The BTP specification was formed to address the needs of inter-organizational transactions and of workflow systems in general. It was also design to overcome the limitations of similar coordination protocols tied to communication protocols.

Another design factors for BTP was to have the new protocol work in conjunction with current business messaging standards, especially those in development by the ebXML Initiative (another OASIS project). That said, BTP is not locked into any one protocol; in fact it can be layered over any transport technology, such as the Simple Object Access Protocol (SOAP) or RosettaNet messaging. BTP also requires that implementations bound to the same carrier protocols should be interoperable. The current specification of BTP describes a SOAP 1.1/HTTP binding.

A key fact to note is that although transaction and security aspects of an application system are often related, the BTP specification consciously does not address how a BTP transaction will integrate with a security system, because Web services security standards were developed independently of the transaction specifications and the WS-Security standards has only just been ratified by OASIS. One of the goals of the BTTC is that BTP

would avoid dependencies on other standards and constraints on implementation choices. This is intended to address a major challenge of B2B development: the problem of how to coordinate the information systems of separate businesses (which typically use different business practices, equipment, and technologies) so that they can communicate effectively. By working independently of any messaging frameworks, complex XML message exchanges among multiple businesses are tracked and managed as ongoing, loosely coupled ‘conversations.’ BTP defines the roles that a business’ software agents (called actors) may perform, the messages that will be exchanged by those actors, and the responsibilities of the actors in those defined roles.

## How It Works

BTP is a protocol, i.e. it is a set of well-defined messages exchanged between the application-systems involved in a business transaction. Each system that participates in a business transaction can be thought of as having two elements—an application element and a BTP element. (Figure 1)

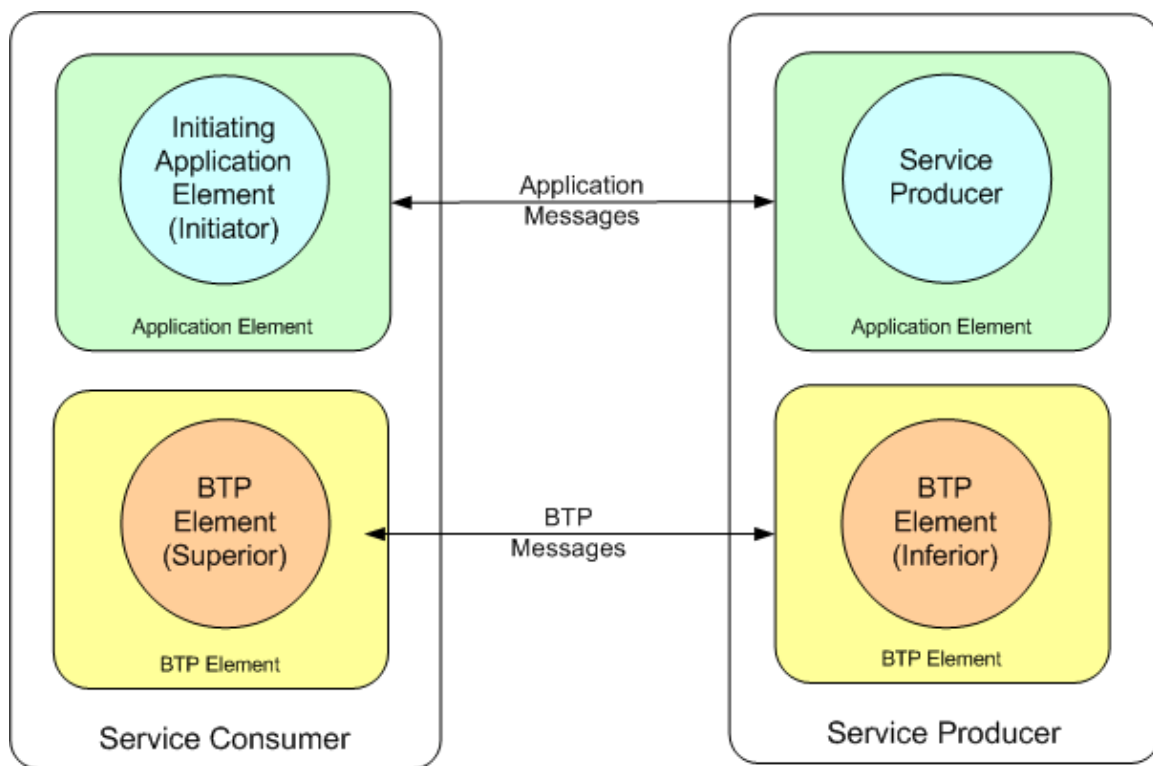


Figure 1: BTP Elements

The application elements exchange messages to accomplish the business function. For example, when you use a money transfer service that sends a message to the banking service with details of the payee's name, address, and payment amount, the application elements of the two services are exchanging a message. The BTP elements of the two

services exchange messages that help compose, control, and coordinate a reliable outcome for the message sent between the application elements.

Note that the application element pertains to the application or business logic that the service consumer and service producer deploy, while the BTP elements are supplied by the BTP vendor. The separation of system components into BTP and application elements is a logical one and these elements may or may not coexist in a single address space.

With respect to a BTP transaction, application elements play the role of initiator (the Web service that starts the transaction) and terminator (the Web service that decides to commit or end the transaction). The initiator and terminator of a transaction are usually played by the same application element.

BTP elements play either a superior or inferior role. The BTP element associated with the application element that starts a business transaction is usually assigned the superior role. The superior informs the inferior when to prepare to terminate the transaction and waits for the inferior to report back on the result of its request. The following parts will discuss the types of transactions and the nature and content of BTP messages.

### **Types of Transaction in BTP**

In traditional transactions, a transaction manager will roll back a transaction if any resource manager participating in the transaction cannot commit or cannot prepare. But in BTP, this cannot be assured. With BTP, you have to define the set of participants that must confirm before a transaction can be committed; this group of participants makes up what is known as the confirm-set. The confirm-set may include all or a subpart of all the participants. To cater for this, BTP has defined 2 types of transactions:

- **BTP Atomic Business Transactions**, or atoms, are like traditional transactions, with a relaxed isolation property.
- **BTP Cohesive Business Transactions**, or cohesions, are transactions where both isolation and atomicity properties are relaxed.

### **Atomic Business Transactions (Atoms)**

Atomic business transaction is where all participants have to agree before a transaction can be committed. If any participant cannot confirm, the entire transaction is canceled. Because BTP transactions do not require strict isolation, it is up to each participating service to determine how to implement transaction isolation. In an atomic Business Transaction, the confirm-set is the set of all inferiors and any of the inferior elements has power to veto the transaction.

Take for example, a web service consumer transaction that invokes two business methods on two different services. If the overall transaction is atomic, the BTP element (superior) at the service consumer end is called a coordinator. In this case, the BTP element plays the coordinator role and coordinates a BTP atomic transaction. It does this by exchanging BTP messages with the BTP elements associated with the two service producers when the application asks it to complete the transaction. On the other side, the inferior BTP elements are called participants and is responsible for persisting the state change made by the associated application element (service producer), which it does by following instructions (via BTP messages) from the superior (coordinator). If either participant informs the superior that it cannot confirm the transaction, the transaction is rolled back.

### **Cohesive Business Transactions (Cohesions)**

Cohesive Business Transactions are transactions where not all involved parties must agree to commit their changes before a transaction is committed. Only some subset of the parties may need to agree. The subset of parties that need to agree before a transaction can be completed is usually determined through the business logic in the application.

In the case of a Cohesion, the BTP element (superior) at the service consumer end is called a composer (as oppose to as a coordinator). The BTP element associated with the service producer is called a participant. In this case, the business logic in initiating application element can determine whether the transaction can be completed, i.e. whether one or both services must confirm. If only one participant must confirm but both eventually confirm, the composer will ask the unwanted participant to cancel or roll back his part of the transaction.

To illustrate this, let's take the example of a travel agent system that would check a few airlines services for the cheapest fare and get the hotel and car rental all in one transaction. In this case we can say that the hotel and car rental can be atomic transaction while for the airlines, only 1 of the airlines need to have a successful booking. We ca say that both the hotel and car rental service are in the confirm-set but only 1 of the airlines need to be in that confirm-set.

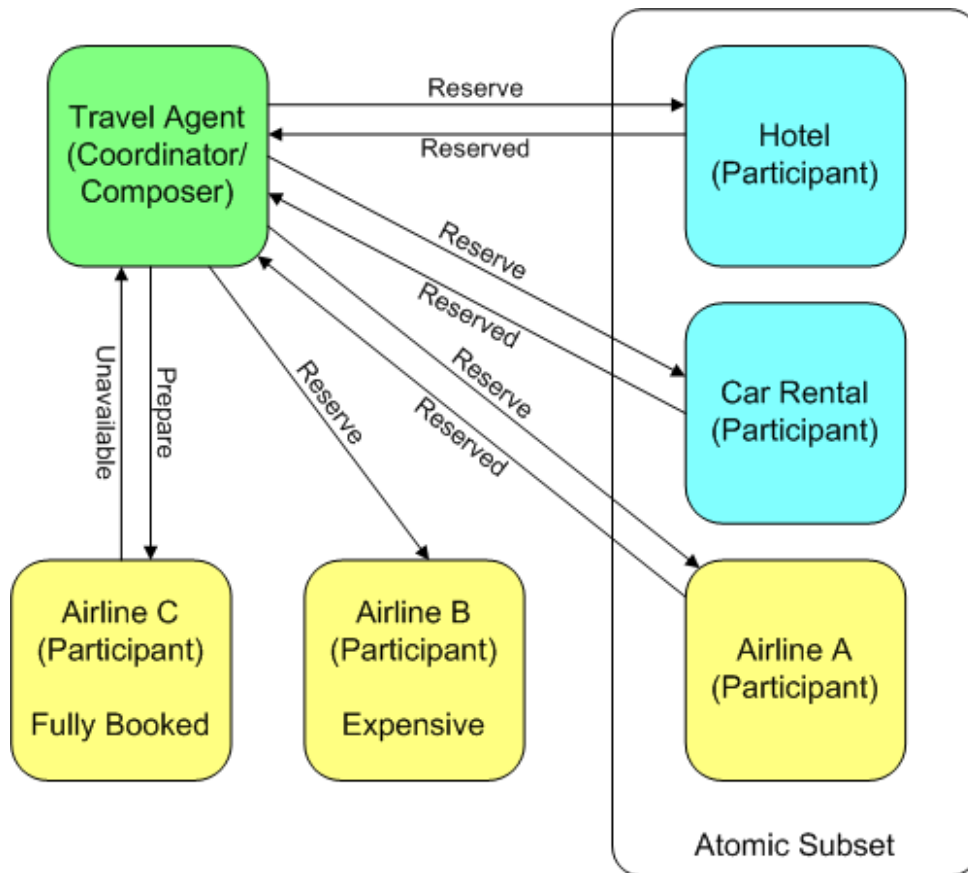


Figure 2 : Transaction Example

Although at first glance, it may seem that the 2 transactions type are distinct, but in fact, cohesions is a superset of atoms: if you have a cohesion coordinator, you can use the same implementation to provide support for atoms, but the inverse is not the case.

In BTP, the actions of transaction coordinator or composer can be influenced by application elements (i.e., business logic). In a cohesion, the initiating application element determines which subset of activities is to be included in the over-all transaction by providing that information to the superior. Application elements can also influence the control and coordination of the transaction by providing the superior with additional context information (via qualifiers; see next section), such as transaction time limits and other application-specific values.

### Locking in BTP

In BTP, for both types of transactions, the isolation level for cohesions is left up to each service. Some of the ways in which applications can achieve isolation include:

- Making changes but applying locks, as in traditional transactions
- Deferring changes until a transaction commits (perhaps by writing a log of changes that will be applied later)

- Making changes and making the interim results visible to others. (This is also known as the provisional effect.)

If a service makes visible provisional changes and the transaction is ultimately rolled back, new compensating transactions may have to be generated to undo the changes made (This is known as the counter effect).

### BTP Players and Messages

The BTP element associated with the initiator plays the superior role and is usually also the *terminator* of the initiated transaction. Depending on the type of business transaction, superiors are either coordinators or composers of the business transaction where an atom is coordinated by an *atom coordinator (coordinator)*, and a cohesion is composed by a *cohesive composer (composer)*. All other BTP elements will be inferiors to this superior. In the simplest case, with only two parties of one superior and one inferior, the inferior is called a participant.

Figure 3 shows a more detailed version of Figure1. The application (*initiator*) first asks a BTP element called the *factory* to create the coordinator/composer. The *factory* creates the superior and returns the transaction *context*. The *initiator* then invokes the business method on the service consumer and passes the context to the service.

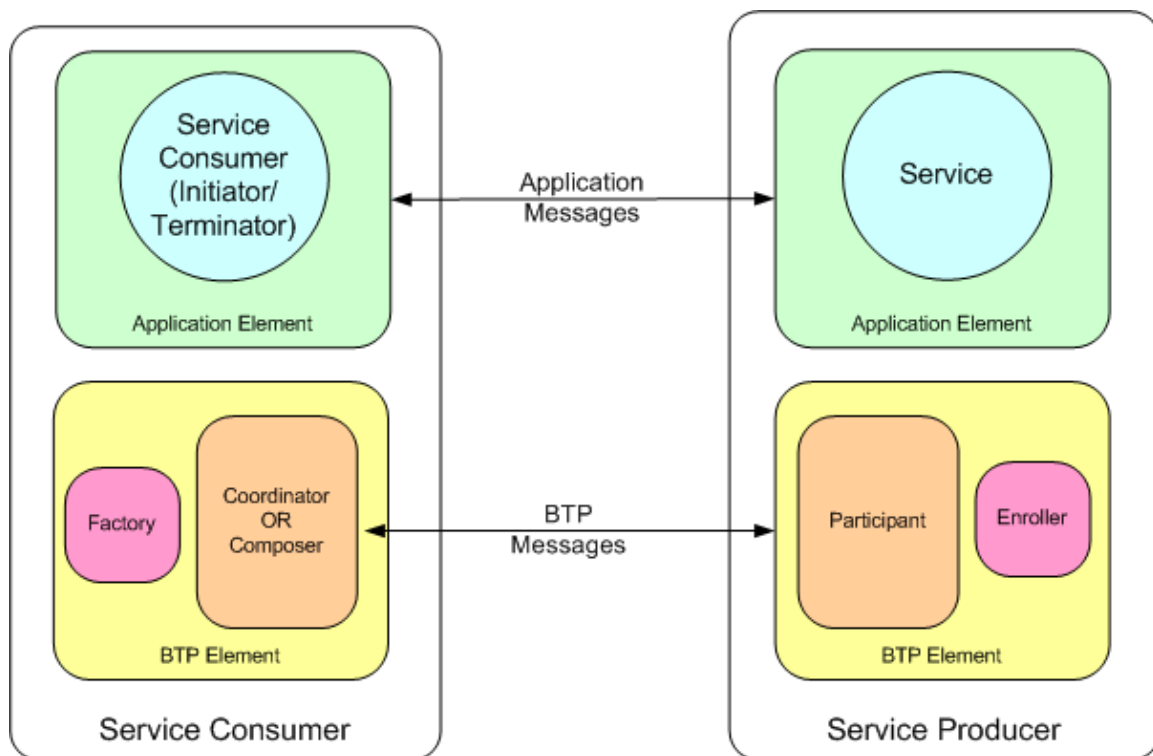


Figure 3: BTP and application elements



How the context is passed depends on the protocol binding; for example, it can be passed as a header block in a SOAP message. At the receiving side, the invoked service asks a BTP element called *enroller* to enroll in the transaction, passing the received context. The enroller creates the inferior (*participant*) and enrolls in the transaction with the superior. Finally, the service provides the response to the business method and passes along the *context reply*. Figure 4 shows the exchange of messages between the different elements.

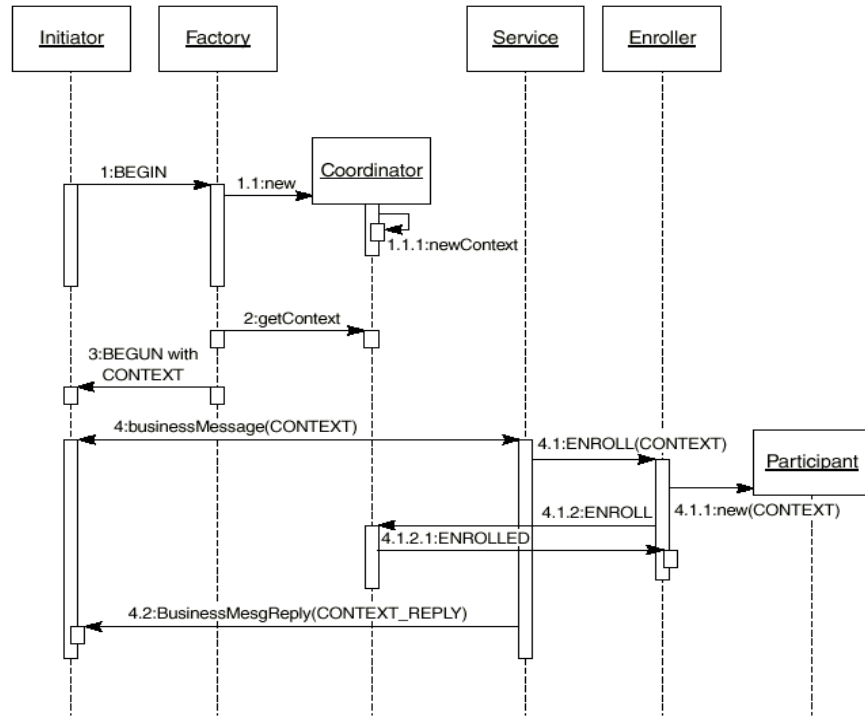


Figure 4 BTP sequence diagram

Note: BTP messages must be bound to a protocol such as SOAP. Because we have not yet described the BTP binding to SOAP, the following section shows only abstract forms of BTP messages

All BTP messages have an associated schema. The CONTEXT message shown below is an example of a BTP message.

```

<btp:context id>
  <btp:superior-address>address</btp:superior-address>
  <btp:superior-identifier>URI</btp:superior-identifier>
  <btp:superior-type>atom</btp:superior-type>
  <btp:qualifiers>qualifiers</btp:qualifiers>
  <btp:reply-address>address</btp:reply-address>
</btp:context>
  
```

The `superior-address` element contains the address to which ENROLL and other messages from an inferior are to be sent. Every BTP address element (`superior-address`, `reply-address`, etc.) has the following XML format:

```

<btp:superior-address>
  <btp:binding-name> </btp:binding-name>
  <btp:binding-address></btp:binding-address>
  <btp:additional-information>information ...
  </btp:additional-information>
</btp:superior-address>

```

`superior-identifier` contains a unique identifier (URI) for the superior. `superior-type` indicates whether the context is for a transaction that is an atom or a cohesion. The `qualifiers` element provides a means for application elements to have some control over transaction management. Qualifiers are data structures whose contents can influence how the transaction coordinator/composer controls the transaction. BTP defines a few standard qualifiers (such as transaction time limit), but BTP vendors can define more such parameters.

The `reply-address` element contains the address to which a CONTEXT\_REPLY message is to be sent (this is required only if the BTP message is not sent over a request-response transport).

### BTP Two-Phase Protocol

The two-phase commit used in BTP is quite similar to the two-phase protocol used in the flat transaction model with some key differences which will be discussed later. Figure 5 shows how such a transaction is terminated using the two-phase protocol.

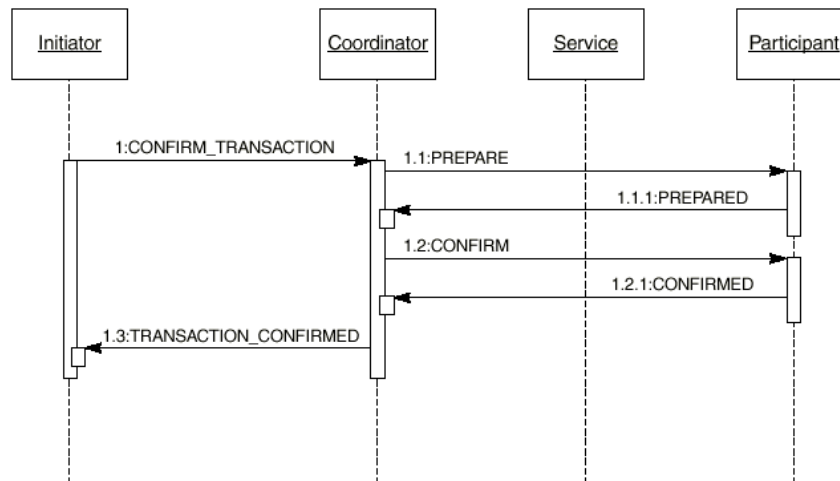


Figure 5 A simple atom example illustrating the BTP two-phase protocol

On receiving a PREPARE message, an inferior (participant) can reply with a PREPARED, CANCEL, or RESIGN message. In Figure 5, because only one inferior exists, the participant must reply with a PREPARED message if the transaction is to be

confirmed and progress to phase 2 (CONFIRM). An example of the BTP message for PREPARE is shown below:

```
<btp:prepare id>
  <btp:inferior-identifier> URI </btp:inferior-identifier>
  <btp:qualifiers>qualifiers</btp:qualifiers>
  <btp:target-additional-information>
    additional address information
  </btp:target-additional-information>
  <btp:sender-address>address</btp:sender-address>
</btp:prepare>
```

The qualifiers element contains a set of standard or application-specific qualifiers. The timeout for inferiors is one of the qualifiers that should be sent for a PREPARE message. *target-address* points to the address of the inferior that was ENROLLED. The PREPARE message will be sent to that address. The *sender-address* points to address of the superior.

The effect on the outcome of a final transaction of having multiple inferiors depends on whether the transaction is a cohesion or is an atom. The set of inferiors that must eventually return CONFIRMED to a CONFIRM message for the transaction to be committed is called a *confirm-set*. For an atomic transaction, the set consist of *all* of a superior's inferiors. For a cohesion, the confirm-set is a subset of all its inferiors. The subset is decided by the application element associated with the superior (this implies that business logic is involved).

Figure 6 illustrates how a composer with multiple participants confirms a cohesion with the two-phase protocol. The application element (the *initiator* and the *terminator* of the transaction) decides that only participants 1 and 2 should confirm—that the *confirm-set* consists of participants 1 and 2. To accomplish this,

1. The terminator sends a CONFIRM\_TRANSACTION with the IDs of the participants in the *confirm-set*.
2. The *decider* (composer) sends PREPARE messages to participants 1 and 2 and a CANCEL message to participant 3.
3. As soon as PREPARED messages return from participants in the confirm-set, the decider sends out CONFIRM (phase 2) messages.
4. When the confirm-set replies with CONFIRMED messages, the transaction is confirmed.

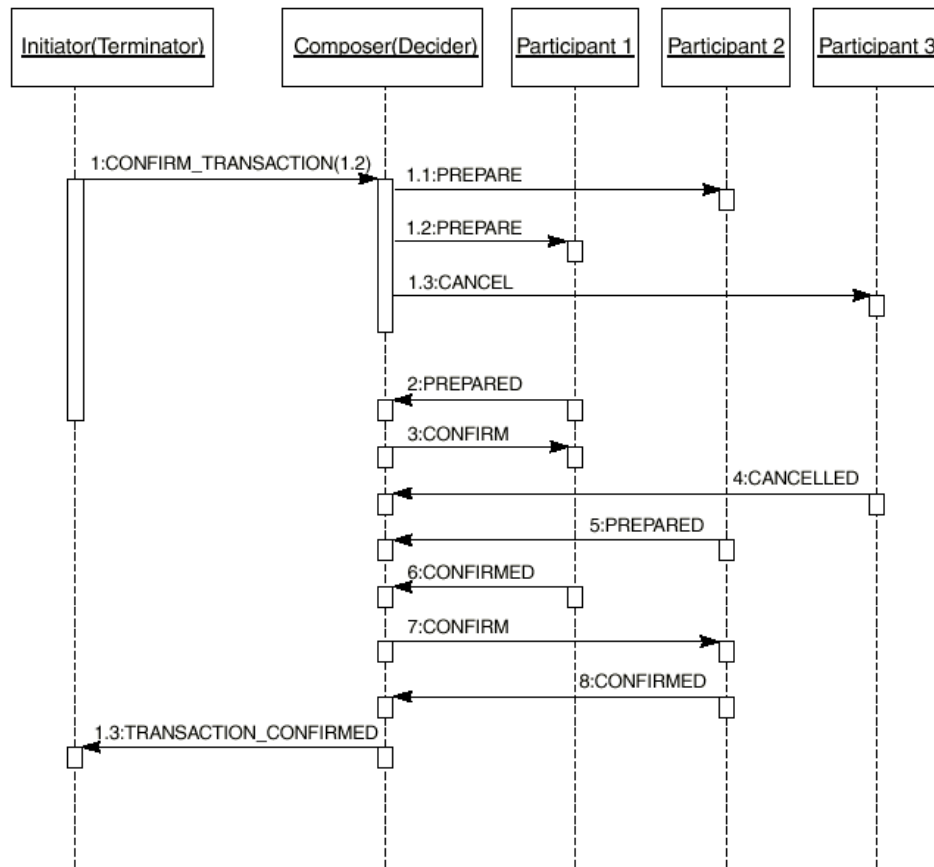


Figure 6: Cohesion completion

How the confirm subset is passed to the decider is better understood by examining the CONFIRM\_TRANSACTION message structure:

```

<btp:confirm-transaction id>
  <btp:transaction-identifier> ... URI ...
</btp:transaction-identifier>
<btp:inferiors-list>
  <btp:inferior-identifier>inferior URI</btp:inferior-identifier>
  <btp:inferior-identifier>inferior URI</btp:inferior-identifier>
</btp:inferiors-list>
<btp:report-hazard>true</btp:report-hazard>
<btp:qualifiers>qualifiers</btp:qualifiers>
<btp:target-additional-information>
  info
</btp:target-additional-information>
<btp:reply-address>decider address</btp:reply-address>
</btp:confirm_transaction>
  
```

Note that the `inferiors-list` contains only the confirm-set of inferiors. If this element is absent, *all* inferiors are part of the confirm-set. For an atom, because all participants are in the confirm set, this element must not be present.

The `report-hazard` element defines when the decider informs the application that the transaction is conformed (TRANSACTION\_CONFIRMED message):

- If `report-hazard` is true, the decider waits to hear from all inferiors, not just the confirm-set, before informing the terminator.
- If `report-hazard` is false, the decider must wait for all elements (even elements that receive a CANCEL message) to reply before communicating the outcome of the transaction to the terminator.

`report-hazard` is useful when the application element wants to know if there was a hazard (problem) with any inferior.

If a coordinator or composer has only one inferior, it may decide to use a single-phase confirm operation and skip the two-phase protocol. Instead of a PREPARE + CONFIRM message exchange, it may send a CONFIRM\_ONE\_PHASE message to the inferior. The two-phase protocol used in BTP ensures that either the entire transaction is canceled or that a consistent set of participants is confirmed.

### Critique of BTP

One of the key advantages of BTP is that it has been worked on for a longer time and is considered well-formed and complete. But although it is pretty straightforward, the volume and complexity can make it hard to digest.

BTP uses business logic to control the flow of the transaction; while this seems to give you more control over how the transaction will flow, it in fact reduces what you would expect from a transaction protocol like consistency, isolation, etc. This means that when using BTP, you would have to do a lot more work to ensure a transaction is valid. Because of this reliance on the business logic to flow the transaction, the user or initiator has to be very close to (or even be) the coordinator. Critical business information such as the ability for a participant to remain prepared (for example, hold onto a hotel room) for a specific period of time is propagated from the participant to the coordinator, but there is nothing within the protocol to allow this information to filter up to the application/client where it really belongs. Because of the lack of flow control in the protocol, in order to use cohesions it is also necessary for Web services to expose back-end implementation choices about participants.

In order to parameterize the two-phase completion protocol, the terminator of the cohesion needs to be able to determine among the participants that has been enrolled in the cohesion, which ones to prepare and which ones to cancel, unlike a traditional transaction system where users do not need to know all the participants. This is something that goes against the Web services orthodoxy. Also, because BTP requires

transaction control to use the open-top approach, it is difficult to leverage existing enterprise transaction implementations. Few transaction systems (or their administrators) will feel comfortable exposing their coordinators through the two-phase interface.

Furthermore, the BTP specification expends great efforts to ensure that two-phase completion does not imply ACID semantics. This is a double-edged sword as it may be good in the sense that traditional ACID transactions are not suitable for all types of Web services interactions, but on the other hand, everything is left up to back-end implementation choices and there is nothing in the protocol (implicit or explicit) to allow a user to determine what choices have been made. Therefore, it is impossible to reason about the ultimate correctness of a distributed application. For example, if you wanted to use BTP for ACID transactions, then of course services could use traditional XA resource managers (for example) wrapped by BTP participants. Unfortunately, there is no way within the BTP for those services to inform external users that this is what they have done so that they can safely be used within the scope of a BTP “ACID” transaction.

## WS-Coordination/WS-Transaction

Before getting into the details of the WS-Transaction, we have to firstly understand WS-Coordination service. The WS-Coordination protocol is used to distributed services. WS-Coordination is an extensible framework for providing protocols that coordinate the actions of distributed applications. Also, it provides a generic framework for specific coordination protocols, like WS-Transaction, to be plugged in.

Traditional transaction protocols assumed that the request and response are synchronous. Where the WS-Transaction is layered upon the WS-Coordination protocol and their communication patterns are asynchronous by default.

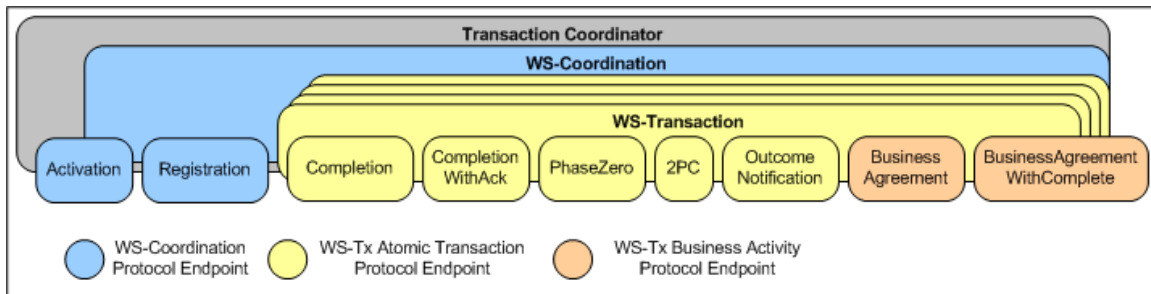


Figure 7: WS-Coordination Foundations

### WS-Coordination:

The foundation of the WS-Coordination has the following functions:

- Activation service that enables an application to create a coordination instance or context. (Option for Coordination Service)
- Registration service that enables an application to register for coordination protocol. (Mandatory for Coordination Service)
- Coordination service type (Stock Trades, Supply Chain) that allows the protocol to be understood by both end of the communication.

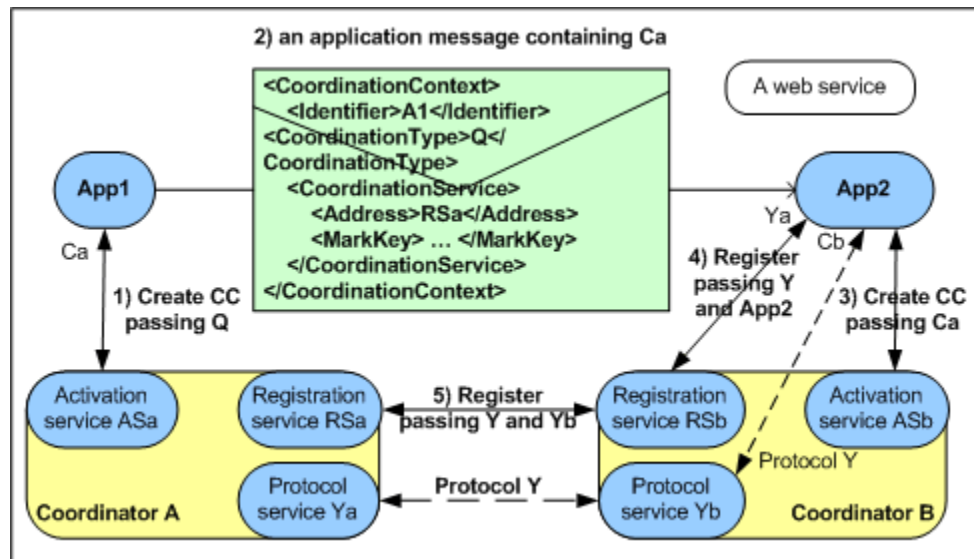


Figure 8: Coordination Service

WS-Transaction specification proposes two distinct models: Atomic Transactions and Business Activity. Both models are extensible and allow implementations to tailor the protocols as they see fit. Below is a context diagram of how WS-Transaction is implemented. (Figure 9)

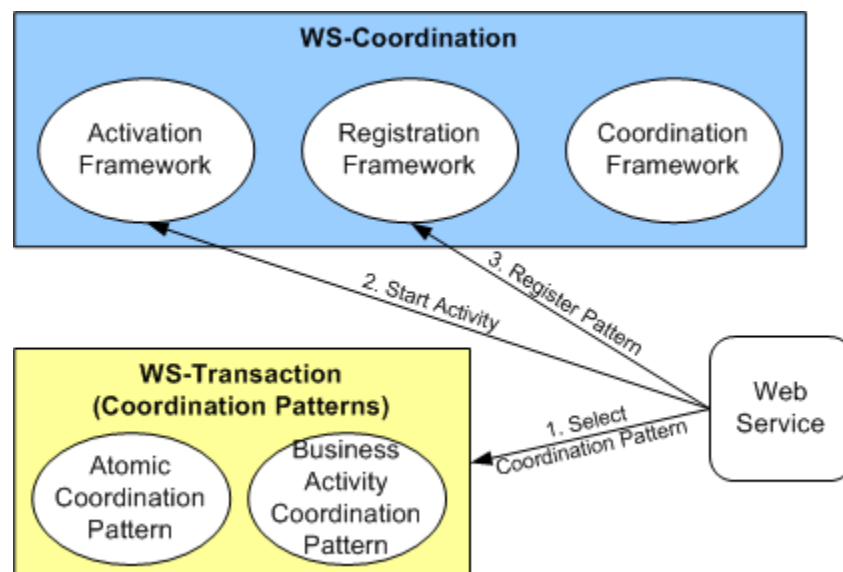


Figure 9: WS-Transaction Context

An atomic transaction is an XML marked-up version of the classical all-or-nothing atomic transaction. It is meant to be used by activities that last for short periods of time, where the locks necessary to maintain consistency will not be held for so long that performance may be degraded. Because of its nature, atomic transactions should be run in a trusted environment to prevent denial-of-service attacks.



How we would determine that a transaction is “short-lived” or not is really subjective and it will differ from architecture to architecture.

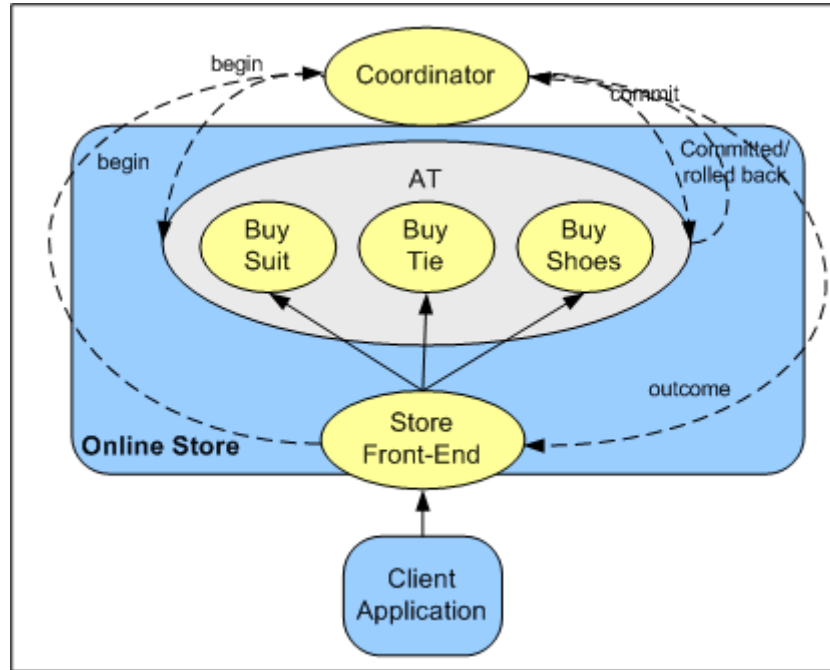


Figure 10: Using AT to ensure All-or-Nothing

### Atomic Transaction (AT) Activities

An atomic transaction consists of five distinct activities:

1. *Completion* - Once the application that created the transaction registers for the completion protocol, it can tell the coordinator to either try to commit the transaction or force a rollback. A status is returned to indicate the final transaction outcome.
2. *CompletionWithAck* - This is basically the same as *Completion*, but the coordinator must remember the outcome until receipt of an acknowledgment notification.
3. *PhaseZero* - The Phase Zero message is sent to interested participants to inform them that the two-phase commit protocol is about to begin. This message allows those participants to synchronize any persistent data to a durable store prior to the actual two phase commit algorithm being executed.
4. *2PC* - A participant registers for these messages for a particular transaction, so that the coordinator can manage a commit-abort decision across all the participants. If more than one participant is involved, both phases of 2PC are executed. If only one participant is involved, a One Phase Commit (a 2PC performance optimization used in the case of a single participant) is used to communicate the commit-abort decision to the participant.

5. *OutcomeNotification* - A participant that wants to be notified of the commit-abort decision registers to receive this “third-phase” message. Applications use outcome notification to release resources or perform other actions after commit or abort of a transaction.

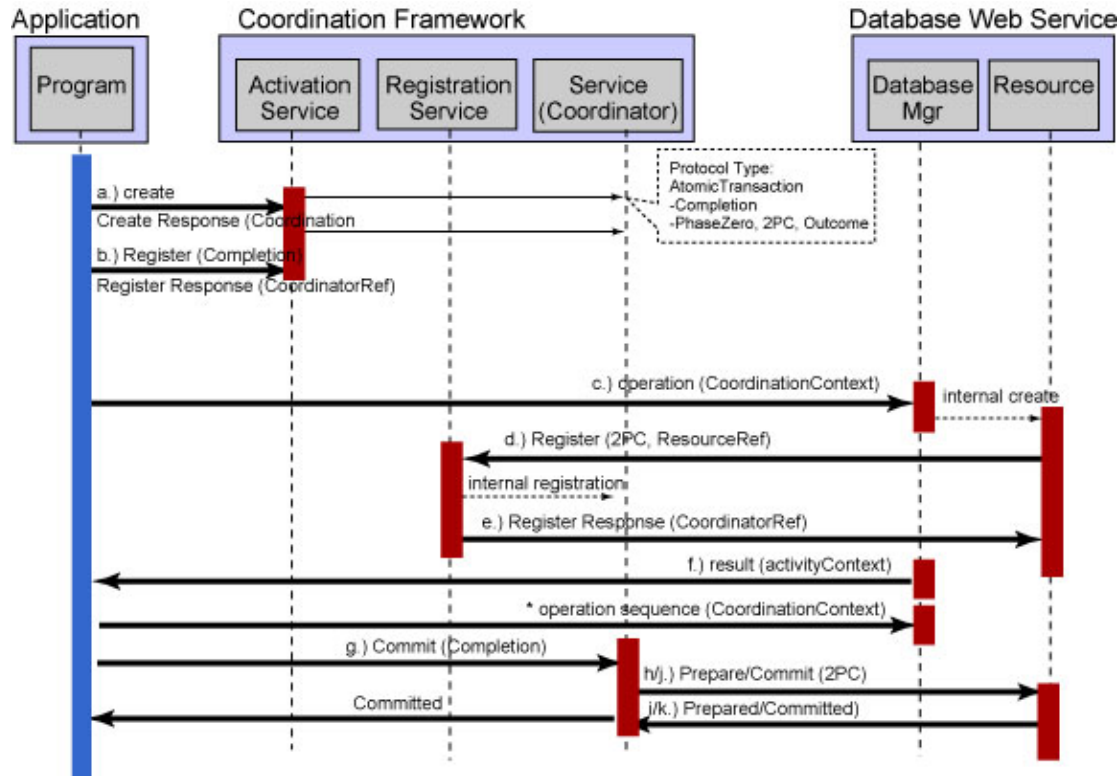


Figure 11: A scenario of Atomic transactions

This example scenario is broken down by each phase of the transaction with details of the steps within each phase. These phases are in turn: *Activation*, *Registration*, and *Completion/Coordination*.

To begin an atomic transaction, the client application firstly locates a WS-Coordination coordinator Web Services that supports WS-Transaction. Once located, the client sends a WS-Coordination “*CreateCoordinationContext*” message to the activation service and will get back an appropriate WS-Transaction context from the activation service.

After obtaining a transaction context from the coordinator, the client application then proceeds to interact with the Web Services to accomplish its business-level work. When all the necessary application level work has been completed, the client can terminate the transaction by first registering its own participant for the “*Completion*” or “*CompletionWithAck*” protocol. After it has been registered, the participant can instruct the coordinator either to try to commit or rollback the transaction. When the commit or

rollback operation has completed, a status is returned to the participant to indicate the outcome of the transaction.

The “*CompletionWithAck*” protocol goes one step further and insists that the coordinator must remember the outcome until it has received acknowledgment of the notification from the participant.

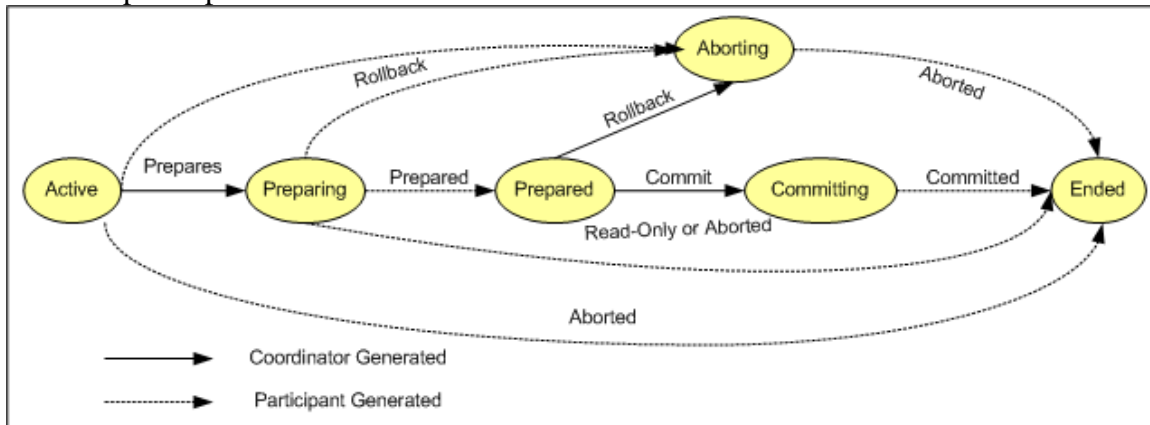


Figure 12: Two Phase Commit State Transition

The two-phase commit protocol is used to ensure atomicity between participants, and is based on the classic two-phase commit with presumed abort technique. During the first phase, when the coordinator sends the prepare message, a participant must make durable any state changes that occurred during the scope of the transaction, such that these changes can either be rolled back or committed later. If the participant cannot prepare them it must inform the coordinator via the “*Aborted*” message and the transaction will ultimately roll back. Assuming no failures occurred during the first phase, in the second phase the coordinator sends the commit message to participants, who will make permanent the tentative work done by their associated services.

### Business Activity (BA) Coordination

BA handles long-lived activities, mostly used in Business-to-Business transaction. Atomic transaction holds resource from multiple parties until being committed or roll backed. Your business partner may not allow you to hold their resources. Some web service connection could be timed out due to system failure or lengthy execution time from your business partners. BA protocol implements business logic to handle such exceptions.

Sample CoordinationContext envelope between two parties:

```

<?xml version="1.0" encoding="utf-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope">
  <S:Header>
    <wscoor:CoordinationContext
      xmlns:wscoor="http://schemas.xmlsoap.org/ws/2002/08/wscoor"
      xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">

```

```

xmlns:myApp="http://www.pace.com/myApp">
  <wsu:Identifier>
    http://Fabrikam123.com/SS/1234
  </wsu:Identifier>
  <wsu:Expires>2004-07-21T13:20:00-05:00</wsu:Expires>
  <wscoor:CoordinationType>
    http://schemas.xmlsoap.org/ws/2002/08/wsba
  </wscoor:CoordinationType>
  <wscoor:RegistrationService>
    <wsu:Address>
      http://Schedule456.com/mycoordinationservice/registration
    </wsu:Address>
    <myApp:Myapp:BetaMark> ... </myApp:Myapp:BetaMark>
    <myApp:EBDCode> ... </myApp:EBDCode>
    <myService:NestedCreate wsu:MustUnderstand="true">
      </myService:NestedCreate>

    </wscoor:RegistrationService>
  </wscoor:CoordinationContext>

</S:Header>
. . .
</S:Envelope>

```

There are basically two types of BA coordination protocols, namely:

1. *BusinessAgreement* Protocol:

With this protocol, the Coordinators send 4 types of messages:

<b>Close</b>	Send by Terminate a business activity normally.
<b>Cancel</b>	Send by Coordinators to back out of a business activity.
<b>Compensate</b>	A message to a Completed scope from a coordinator to execute its compensation.
<b>Forget</b>	Send by coordinators when received faulted message from participants.

The Participants send 6 types of messages (Same as *BusinessAgreement* Protocol):

<b>Completed</b>	Participant notifies the coordinator when the participant finished the tasks but waiting for close or compensate messages from coordinator.
<b>Faulted</b>	Participant failed to execute or compensate the transactions.
<b>Compensated</b>	Successfully compensated the transactions as requested by coordinator.
<b>Closed</b>	Participant replies to close request from the coordinator.
<b>Canceled</b>	Participant replies to cancel request from the coordinator.
<b>Exited</b>	Send by participant when the participant finishes all the tasks and the nature of the task requires no more participation in the business activity.

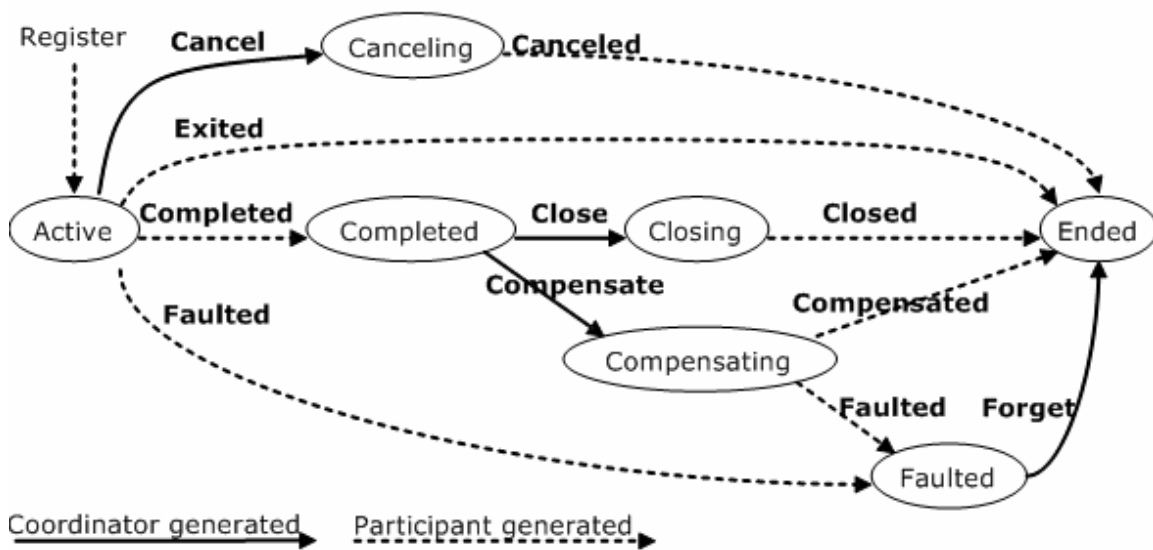


Figure 13: BusinessAgreement Protocol State Diagram

## 2. *BusinessAgreementWithComplete* Protocol:

With this protocol, the Coordinators send 5 types of messages:

<b>Complete</b>	Send by coordinators to participants when participants received all the required transactions from coordinators.
<b>Close</b>	Send by Terminate a business activity normally.
<b>Cancel</b>	Send by Coordinators to back out of a business activity.
<b>Compensate</b>	A message to a Completed scope from a coordinator to execute its compensation.
<b>Forget</b>	Send by coordinators when received faulted message from participants.

And the Participants send 6 types of messages (Same as *BusinessAgreement* Protocol):

<b>Completed</b>	Participant notifies the coordinator when the participant finished the tasks but waiting for close or compensate messages from coordinator.
<b>Faulted</b>	Participant failed to execute or compensate the transactions.
<b>Compensated</b>	Successfully compensated the transactions as requested by coordinator.
<b>Closed</b>	Participant replies to close request from the coordinator.
<b>Canceled</b>	Participant replies to cancel request from the coordinator.
<b>Exited</b>	Send by participant when the participant finishes all the tasks and the nature of the task requires no more participation in the business activity.

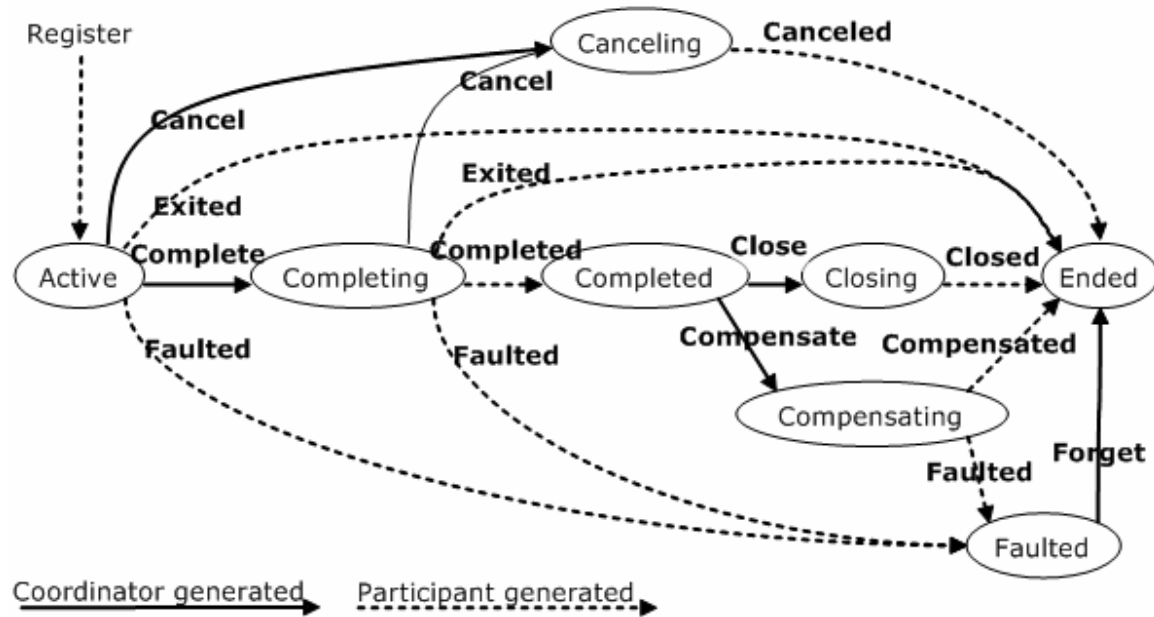


Figure14: BusinessAgreeWithComplete Protocol State Diagram

### Compare and contrast Atomic and BA protocol:

	Atomic Protocol	BA Protocol
<b>Average Execution Time</b>	Short	Long
<b>Scope</b>	Mostly Internal Systems	interoperates with multiple external systems
<b>Resource Locking</b>	Lock the resource. Prevent changes from other transactions	Don't lock the resource. Flexible isolation policies or compensations.
<b>Roll Back</b>	Abort transaction	Use compensation to reverse the effects of the original business task.
<b>Request Time out</b>	Abort transaction and retry	Resent request

Table 2: Difference between Atomic and BA protocol

### Critique of WS-Coordination/Ws-Transaction

With WS-C/WS-Transaction, applications have to communicate through coordination services, which could exist outside of company firewall. Domain-specific coordination protocols have to be created and inserted into coordination services and out of the box WS-Coordination service provides only activity and registration services. Another point to note is that most business logics are defined in the coordination protocol services within the coordinator, so there is less flexibility to change the business logics.

## Comparative Analysis

Although there is commonality between the two specifications (both support a two-phase completion protocol, for example) there are many more differences between the 2 models, the key differences can be categorized into 2 general points:

BTP	WS-Transactions
BTP was not specifically designed just for Web Services; it can be used for other environments. As such, BTP defines the transactional XML protocol and must specify all of the service dependencies within the specification.	WS-C and WS-Transaction are specifically designed for the Web Services environment and hence build upon the basic definition of a Web service infrastructure.
BTP does not assume any transaction infrastructure, and thus has to essentially start from scratch and requires business-level decisions to be incorporated within the transaction infrastructure.	Foundations of WS-Transaction are based upon traditional transaction infrastructures, where there is a strong separation between the functional aspect of business logic and the non-functional aspects of using transactions within an application.

*Table 3: Key Differences between Models*

Some of the major difference can be seen in the way each model defines their semantics of the protocol. For example, each model in WS-Transaction clearly defines the semantics within the protocol (Atomic Transaction is ACID, for example), this is because the models in WS-C/WS-Transaction are each aimed at a specific problem domain and is not intended to be used as a global panacea. On the other hand, BTP does not have such well-differentiated models; the cohesion model is essentially a superset of the atom model, thereby limiting itself to only 1 model to solve all problems. The differentiator for BTP, while not in the semantics, is in the boundaries of the properties of ACID. By relaxing the restrictions on properties such as atomicity and durability within the protocol, it allows those semantics to be defined outside of the model. This approach can be considered a “double-edged sword” as on one hand, it gives great flexibility to the application developer, but on the other hand, it does not allow them to be able to reason about an application’s overall functionality and behavior, thus making it very difficult to construct applications from arbitrary services since within the protocol.

As can be seen from the table below, the differences between the 2 protocols are many but subtle. There are also similarities between the two protocols; both WS-C/WS-Transaction and OASIS BTP can be used to support business process execution environments like BPEL4WS, WSFL, WSCI, BPML, and others which make both models useful as an implementing technology for things like workflows; and in terms of types of transactions, the high level mechanism of WS-C/WS-Transaction Business Activities is very similar to BTP Cohesions.

	<b>BTP</b>	<b>WS-Transaction</b>
<b>Coordination framework</b>	None, tied to 2-phase	WS-Coordination
<b>Transaction framework</b>	General protocol, statically defined	None, but current defined protocols cover typical patterns (AT and BA)
<b>Strict atomic model</b>	Atom, which is atomic only, other properties specified by service (not available via protocol). Uses open-top protocol which makes interoperability with existing transaction systems difficult.	Atomic Transactions, which requires strict ACID properties, specifically for interoperability with traditional transaction systems.
<b>Relaxed model</b>	Cohesion allow flexible participant list. Requires participants to be exposed to application/terminator	Business Activity allows flexible participant list.
<b>Scopes</b>	No. Cohesion manages relationship within scope.	Yes, Business Activity manages relationship between scopes. Nested scopes allowed.
<b>Flexible outcomes for consensus groups</b>	Yes, via Cohesions	Yes, via Business Activity.
<b>Flexible participation in consensus groups</b>	Yes, participants can resign from Cohesion.	Yes, participants can exit in Business Activity protocol
<b>Service behavior</b>	Services define behavior (not specified by BTP)	Defined by the protocol
<b>Business logic/coordinator separation</b>	Mixed (open-top protocol requires strong coupling between business logic and coordinator)	Distinct
<b>Web services-specific</b>	No, requires a lot of extra effort from the specification/protocol	Yes
<b>Failure recovery</b>	Re-drive protocol	Optimized protocol

*Table 2: Summary of similarities and differences*

## Conclusion

In our review, we have found that both models are relevant and can be used in many of today's transactional context.

BTP was developed to solve what was then deemed as a new problem beyond traditional transaction support, but after implementation, it was found that it required brand new architecture and infrastructure to support it. In some sense, it defeats the purpose of having Web Service as a connector across company domains mainly because with BTP, back end implementations of participating companies needs to be exposed for BTP to operate successfully. In many established firms (such like finance, brokerage and healthcare firms) that have heavy security concerns, exposing the back end



implementation is not something that would be readily acceptable. In those cases, BTP might not be an option.

As for WS-transactions, the infrastructure to implement it is more extensive as you would also require a coordination service which could incur higher cost.

BTP allows finer control and flexibility of implementing business transactions. In cases where the transaction logic and participants are simple, BTP offers a more attractive approach.

Our team's recommendation is to examine the nature of the business transactions and the complexity of the transaction workflow together with security and infrastructure concerns of the organization before determining the right fit. We believe that both models have their purposes and will be useful in their own ways depending on environment.

## References

McGovern, J., Tyagi, S., Stevens, M., Mathew, S., (2003). Java Web Services Architecture. Morgan Kaufmann.

OASIS Business Transactions Technical Committee

[http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=business-transaction](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=business-transaction)

Specification: Web Services Transaction (WS-Transaction). August 2002

<http://www-106.ibm.com/developerworks/library/ws-transpec/>

Web Services Coordination (WS-Coordination). September 2003

<http://www-106.ibm.com/developerworks/library/ws-coor/>

WS-Transaction Specification Index Page

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/wsatspecindex.asp>