Web Service Transactions

An evaluation of description- and management- standards and languages

Technical Report

Chang-ai Sun
Dieter Hammer
Gerard Biemolt
Heerko Groefsema

Version: 1.0
Date: 14 July 2006
1 Introduction

1.1 Web Services Overview

Web services form a wide range of web applications that are written in several different languages. Some systems are possible not replaceable or it is too expensive to transfer to a new standard. The Web Services Description Language (WSDL) provides an extra layer upon all web services. There is a new level of abstraction on which the WS-* provides specifications for the business processes. These specifications form the WS-stack, which can be seen in figure 1 [DUST06]. Specifications are available on all levels ranging from the business process integration down to the messaging and transport protocols.

![Web Service Stack](image)

**Figure 1: Web Service Stack.**

WS-CDL and BPEL operate on the highest level and form the composition specification and implementation for the web services. Quality of service is assured by implementation of the specifications such as WS-Security and WS-ReliableMessaging. WS-Coordination and WS-Transaction, which are subject of this report, are also on the Quality of service level, but are directly related to the transactions management. WS-Coordination sets up the connection and transportation protocol between web services and WS-Transaction specifies a series of communication protocols. The lower layers hold the WS-Policy and messaging protocols that take care of the actual transport of the messages.

This document provides an analysis and evaluation of the current standards and languages for the description and management of web based transactions. We first introduce the evaluation criteria to create the possibility for a clear-cut one-on-one evaluation. This evaluation will then follow by combining and comparing each component and standard. The ultimate goal is
to provide a comparison between the standards and languages such that it can be used to make a rational choice. Such a choice will be based on the possibilities and limitations which need to be investigated in this report.

The evaluation criteria consist of the requirements and an example. The transaction requirements have been formed by common knowledge and research into the area of transactions by studying literature on transaction management and specifications of the standards, but also by regular meetings with scholars. The standards are selected upon their applicability on management transactions in service centric systems.

The first section holds the transaction requirements, which are illustrated by an example, it contains motivations for each and every requirement. The second section holds summaries of the available standards and languages on Transactions. The next section holds summaries of the available standards and languages on Compositions. In both sections the example will be applied to the standards to see the features that are supported and, even more important, the features that are not supported. Finally the last section holds a summary of the third and fourth section where the observations are listed and explained.

1.2 Transactions Overview

Services communicate via messages. A sequence of messages can be bound into a transaction in order to achieve certain consistency constraints. A transaction can be defined as a mechanism for ensuring that all participants in an application achieve a mutually agreed outcome [DUST06].

The transactions between web services can also be composed into more complex transactions where several participants perform a composite transaction. These participants are web services itself, but also form a new more complicated web service. Those composite web services can again be invoked by a different web service.

1.3 Choreography versus Orchestration

There are several differences between Choreography and Orchestration. Choreography is a model encompassing all parties and their associated Interactions, while in Orchestration web service coordinates the interactions. Thus Orchestration is based on central control, while Choreography is based on distributed control. In an Orchestration web services do not need to know about each other because the coordinator takes care of the control. In a Choreography however, they do need to know what to do with whom and when.

From the perspective of composing web services orchestration is a more flexible paradigm and has the following advantages over choreography [MATJ]:

- The control flow is simpler and less error-prone since the coordination of web services is centrally managed by a known coordinator.
- Web services can be incorporated without being aware that they are taking part in a larger business process.
• Alternative scenarios can be easier implemented if rebinding or replanning needs to take place, e.g. in case of faults.

<table>
<thead>
<tr>
<th></th>
<th>Orchestration</th>
<th>Choreography</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Coordination</strong></td>
<td>Central</td>
<td>Distributed</td>
</tr>
<tr>
<td><strong>Adaptability</strong></td>
<td>Relatively easy</td>
<td>More complex</td>
</tr>
<tr>
<td>Interaction knowledge</td>
<td>Kept centrally (coordinator)</td>
<td>Must be distributed</td>
</tr>
<tr>
<td>More suited for</td>
<td>Dynamic environments</td>
<td>Static environments</td>
</tr>
<tr>
<td><strong>Transaction standards</strong></td>
<td>BPEL</td>
<td>WS-CDL</td>
</tr>
</tbody>
</table>

Table 1: Comparison of the orchestration and choreography paradigm.

1.4 Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].
2 Evaluation Criteria

As we have mentioned earlier, we shall be evaluating the current Transaction related standards. To do this we need evaluation criteria and good supporting material. We shall first describe the example, and will then go into the transaction requirements. The standards will be evaluated against a set of requirements. A running example will be used for explanatory purposes.

2.1 Example: Drop dead order

A drop-dead order example is illustrated in figure 2. The example is taken from [MPEBT02].

A customer places an order at a distributor for a product and the shipment of it. The Distributor can only take the order if he has the confirmation from both a supplier and a carrier for the supply and delivery of the product. If he can’t get confirmation from both a Supplier and a Carrier, the transaction is aborted and compensating actions have to be taken. If both the Carrier and Supplier are assigned, the Distributor takes the order.

![Use Case diagram of the example scenario](image)

Figure 2: Use Case diagram of the example scenario [MPEBT02].
2.2 Requirements

To get a better feel of transactions we should define them in a proper way. For convenience we have divided the requirements into the ACID (Atomicity, Consistency, Isolation, Durability) properties. The reasoning behind the requirements will be given below.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Must</th>
<th>Should</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Atomicity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>Atomicity is supported.</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>Rollback is supported.</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>Compensating actions are supported.</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Consistency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>Consistency is supported.</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td>A (composite) Transaction is able to abort.</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>Adding deadlines to Transactions is supported.</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>Logical expressions for specifying constraints are supported.</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Isolation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.0</td>
<td>Isolation is supported.</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Durability</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0</td>
<td>Durability is supported.</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Miscellaneous</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>Composite Transactions are supported.</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>5.2</td>
<td>Distributed Transactions are supported.</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>5.3</td>
<td>Transaction recovery by dynamic rebinding and dynamic re-composition at runtime is supported.</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>5.4</td>
<td>Secure Transaction of different types can be established. (Confidentiality, Integrity, Authentication and Non-repudiation)</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>5.5</td>
<td>Optimistic or pessimistic concurrency control is supported.</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Requirements.

1.0 Atomicity is supported

**Meaning:** Atomicity is the ability of a Transaction to either succeed successfully or not at all, even in the event of failure.

**Example:** It should not happen that the supplier’s resources are committed and the Carrier is not.

1.1 Rollback is supported.

**Meaning:** In the event of a failure during a Transaction, all changes made during the process so far should be undone to enforce consistency by rolling back into the initial state.

**Example:** When the Distributor assigned a Supplier but can not assign a Carrier, the changes made with the Supplier (and Customer) should be rolled back.

1.2 Compensating actions are supported.
**Meaning:** In the event of a failure during a Transaction, all changes made during the process so far should be undone to enforce consistency by compensating to a state close to the initial state.

**Example:** When the Distributor assigned a Supplier and committed it but can not assign a Carrier, the changes made with the Supplier (and Customer) should be undone.
2.0 Consistency is supported.
**Meaning:** The ability of a transaction to begin and end in a legal state and thus it not breaking any integrity constraints.
**Example:** When the Distributor assigned a Supplier but can not assign a Carrier, the Transaction with the Supplier should be rolled back and not persist. Otherwise only half the Transaction is completed.

2.1 A (composite) Transaction is able to abort.
**Meaning:** A Transaction should be able to abort and return to the initial state in case of failure or if the user wishes so.
**Example:** When the Distributor assigned a Supplier but can not assign a Carrier, the entire Transaction should be aborted.

2.2 Adding deadlines to Transactions is supported.
**Meaning:** A Transaction might have one or more deadlines.
**Example:** You could imagine that the Customer needs the deliverables before a certain time, which means that the whole Transaction should be done at an early enough time so the Distributor and Carrier have the time to process it. If the Transaction would stay open for too long, it would mean that the schedule wouldn’t be made.

2.3 Logical expressions for specifying constraints are supported.
**Meaning:** Consistency constraints are defined by logical expressions.
**Example:** It should not occur that the account of the Distributor is debited while the account of the Customer isn’t credited in the event of a money exchange.

3.0 Isolation is supported.
**Meaning:** Isolation is the ability of a transaction to perform operations isolated from all other operations. One Transaction can therefore not see the other Transaction’s data in an intermediate state.
**Example:** The Customer should not be aware of the state of the transaction between the Distributor and the Supplier/Carrier.

4.0 Durability is supported.
**Meaning:** Durability is the ability of a transaction to record the results in a persistent way. Whenever a transaction notifies one participant on success, it must persist, even when errors occur. If this wouldn’t be the case, inconsistencies would arise due to the fact that one participant might think that the transaction succeeded while the other may not believe so.
**Example:** When the Supplier is notified on success, but somehow the connection with the Carrier fails, the changes with the Carrier should still be made.

5.1 Composite Transactions are supported.
**Meaning:** A composite transaction is a Nested Transaction.
**Example:** In our example we see a composite transaction; it consists of three smaller transactions that all depend on the global outcome, which is either that all three succeed or that the whole composite transaction fails in case of one or more failures.
5.2 Distributed Transactions are supported.  
Meaning: A distributed transaction is a transaction between two or more services executing on different hosts. The transaction should support transactions through a network between two different hosts.  
Example: A customer can place a drop-dead order at Distributor through a network connection.

5.3 Transaction recovery by dynamic rebinding and dynamic re-composition at runtime is supported.  
Meaning: Dynamic Rebinding is the possibility of forming a new binding at runtime with a different service when the current service is not able to fulfil its promises. The new service must have the same functionality and QoS characteristics as the failing service. Dynamic Re-composition is the forming of a new composition by replacing one or several services by another composition that fulfils the same function.  
Example: We can imagine that the first Carrier somehow fails and is unreachable. If this would happen during a transaction, it should automatically re-bind with a service that offers the same service. It could also re-compose through re-binding with a third Carrier through the Supplier, which would create a whole new composition of the transaction.

5.4 Secure Transaction of different types can be established. (Confidentiality, Integrity, Authentication and Non-repudiation).  
Meaning: Participants in a Transaction can be authorized and authenticated. Data integrity should be maintained. Also, mutual agreements can not be denied afterwards.  
Example: You could imagine that you would want a Transaction to be secure. Both an authentication and authorization and encryption protocols should be supported by the Transaction mechanism.

5.5 Optimistic or pessimistic concurrency control is supported.  
Meaning: A Transaction must support concurrency control to enforce consistency. This control could either be optimistic or pessimistic, but should be optimistic for better performance.  
Example: When two transactions are made concurrently, they should not both claim the same supply of goods from one Supplier.
3 Transactions Standards

In this section we will describe the available standards on transactions. We will give a short introduction into each standard. We then will go into more detail and describe what the standards can add to transaction description or transaction management. Furthermore we will discuss the details and a running example. Finally, we will evaluate the standards against the requirements in section 2.

3.1 BTP


**Abbreviation:** Business Transaction Protocol

**Organization:** OASIS

**Language(s):** XML

3.1.1 Overview

The Business Transaction Protocol (BTP[BTP02]) is an XML-based protocol being developed by the Business Transactions Technical Committee (BT TC) of the Organization for the Advancement of Structured Information Standards (OASIS) as a standardized Internet-based means of managing complex, ongoing business-to-business (B2B) transactions among multiple organizations. The protocol is intended to be especially useful in a Web services environment. The BT TC criteria specify that the protocol works in conjunction with current business messaging standards, especially those in development by the ebXML Initiative. BTP can be implemented on top of any transport technology, such as the Simple Object Access Protocol (SOAP), RosettaNet, or ebXML messaging. BTP works independently of specific transport protocols and 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.

3.1.2 Details

![Figure 3: Superior-Inferior relation (BTP02).](image-url)
BTP defines a BTP Element next to the Application Element or service (figure 3 [BTP02]). This BTP Element contains the Transaction logic for either a role as Superior, Inferior or both (Appendix B [BTP02]). The BTP Element of the invoking Application Element will be Superior and the BTP Element of the invoked Application Element Inferior. A Superior can have one or more of such inferiors, but an Inferior can only have one Superior. For nested transactions, an Inferior itself can also have inferiors, which makes it a Superior as well. An Inferior that isn’t also a Superior is also called a Participant. These relations and elements can be seen in Figure 4, where the squares represent Application Elements and circles BTP Elements.

A Superior can treat its Inferior in two different ways, atomically or cohesively. For cohesive behaviour, the Superior can allow some Inferiors to Cancel while others Confirm. This is not allowed for atomic behaviour, in which all participants must Confirm or all must Cancel. This results in the following four roles in which a superior can act:

- **Atom Coordinator**
  The Application Element initiated a Business Transaction (causing the creation of the Superior), and instructed the Superior that all Inferiors should Confirm or all should Cancel.

- **Cohesion Composer**
  The Application Element initiated a Business Transaction, but deferred the choice of which Inferiors should Confirm until later, allowing it (the Application Element) to choose some subset to be confirmed (the confirmset), others to Cancel.

- **Sub-Coordinator**
  The Application Element was itself involved in a running Business Transaction, and the Superior in this relationship is the Inferior in the invoking relationship. The Application Element in question instructed that all Inferiors of this Superior should Confirm or cancel, but only if this is instructed from above.

- **Sub-Composer**
  The Application Element was itself involved in a running Business Transaction, and the Superior in this relationship is the Inferior in the invoking relationship. The Application Element in question deferred the choice of which Inferiors should be candidates to Confirm until later, allowing it (the Application Element) to choose some subset to be confirmed (the confirmset), given that confirmation is instructed from above, others to Cancel.
As seen in figure 5, the transaction can be separated in three parts, beginning, enrolling and the transaction itself. Beginning starts with the initiator, which contacts the other services, the Factory in the case of figure 5, by a BEGIN message with the request for a coordinator or composer. The Factory requests the transaction context from the BTP coordinator/composer element. The context and the confirmation of the BEGIN (and its response; BEGUN) message are sent to the initiator. The initiator now has addressing and identification information in the form of the context.

The initiator sends the request for service with the context to the service it wants to interact with. The service enrols at the coordinator by the ENROL/ENROLLED BTP messages. The service then replies to the initiator.

Finally, when the communication context is agreed upon, the transaction itself, between the initiator, coordinator and participant can take place. The initiator requests transaction confirmation at the coordinator. The coordinator in turn checks if the service can fulfil the transaction by the PREPARE/PREPARED messages. The service can be confirmed upon positive feedback, the CONFIRM/CONFIRMED messages are then sent. Finally the coordinator sends his confirmation message to the initiator, which completes the transaction.
Figure 5: A conventional BTP message sequence for a simple transaction [BTP02].
3.1.3 Example

Figure 6: Drop Dead Order Example in BTP.

Figure 6 shows an overview of the elements and roles for the example introduced in Section 2.1. In this figure we can see the BTP Elements (the circular objects) next to the Application Elements or Services. In this case the Customer Application Element (Customer) is the Initiator of the Transaction and, because the Customer only connects to one Distributor, its BTP Element is an Atom Coordinator. If the Customer had connected to more than one Distributor, it could have made a choice between them in its confirmset. In that case its BTP Element would have been a Cohesion Composer.

The BTP Element of the Distributor is a Sub-Coordinator, which means it has a Superior in the form of the Customer’s BTP Element and has Inferiors of its own in the form of the two Participant BTP Elements of the Carrier and Supplier. Again, if the Distributor had connected to more Inferiors of the same kind, it could have made a choice between Carriers and Participants in its confirmset. In this case its BTP Element would have been a Sub-Composer instead.

3.1.4 Requirements Analysis

1.0 Atomicity is supported.
Atomicity is supported through the use of the 2PC protocol in the confirmset.

1.1 Rollback is supported.
Rollback is supported through the Cancel message of BTP’s state machines (Appendix B).

1.2 Compensating actions are supported.
Compensating actions are not supported by BTP, because the state machines (Appendix B) do not support Undo messages after changes have been made.

2.0 Consistency is supported.
Consistency is supported through the state machine’s request-response message couples (Appendix B). Each message has to be confirmed by the other party, before a state change is made.
A (composite) Transaction is able to abort.
A (composite) Transaction is able to abort by the Cancel message. The Initiator/Terminator also has to confirm the transaction in the end, and can then terminate the Transaction if needed (Figure 5). In the example described in figure 6, the Customer fulfils this function.

Adding deadlines to Transactions is supported.
Adding deadlines to Transactions is not supported. An Application Element however can simply choose to drop the Transaction when the Transaction takes too long.

Logical expressions for specifying constraints are supported.
Logical expressions for specifying constraints are not supported

Isolation is supported.
Isolation is not supported because it is considered a local matter. An Application Element should therefore decide about Isolation on itself [BTP02].

Durability is supported.
Durability is supported through the states of BTP, which require a Confirmed reaction to the Confirm message sent by the BTP Element (Appendix B). Once the Committed message is returned, BTP states that all changes must be saved by all parties.

Composite Transactions are supported.
Composite Transactions, such as the one in our example are supported. A Superior can have more than one Inferior to register for the Transaction, and can also be an Inferior himself (e.g. the Distributor his BTP Element in the example).

Distributed Transactions are supported.
Distributed Transactions are supported because BTP is build upon SOAP, which again is built upon XML and HTTP and hence supports Distributed Transactions.

Transaction recovery by dynamic rebinding and dynamic re-composition at runtime is supported.
Dynamic rebinding and dynamic re-composition is not supported because it is considered an implementation issue. It is however possible to create a confirmset of the set of Inferiors that were involved in the Transaction. This makes it possible for the Application Element to choose which Inferiors should commit, and which should be cancelled. Transaction recovery through dynamic rebinding is specified in case one party fails and reconnects again [BTP02].

Secure Transaction of different types can be established. (Confidentiality, Integrity, Authentication and Non-repudiation).
Secure Transaction are not supported by BTP itself. Security can however, be supported by using WS-Security in cooperation with SOAP.

Optimistic or pessimistic concurrency control is supported.
Concurrency control is not supported because it considered an implementation issue. Also the state charts (Appendix B) describe only isolated transactions.
3.2 WS-T

**Abbreviation:** Web Service-Transactions  
**Organization:** IBM, BEA Systems, Microsoft, Arjuna, Hitachi, IONA  
**Language(s):** XML

### 3.2.1 Overview

WS-Transactions (WS-T) are specifications developed by IBM, BEA Systems, Microsoft, Arjuna, Hitachi and IONA. They define mechanisms for interoperability between different Web Services and provide a way of composing transactional qualities of Web Services. The WS-T specification is built upon an extensible coordination framework and describes two coordination types. The coordination framework (WS-Coordination) enables existing transaction processing, workflow, and other systems to hide their proprietary protocols and to operate in a heterogeneous environment. The two coordination types, WS-AtomicTransaction (WS-AT) and WS-BusinessActivity (WS-BA), each define a number of agreement protocols that can be implemented within the framework. The difference between the two is that the first describes short duration ACID transaction protocols, while the latter describes longer running business transactions that are not necessarily ACID but use compensating actions.

### 3.2.2 Details

First we give the details of the slightly higher levelled WS-Coordination, which specifies the coordination framework for both WS-AtomicTransaction and WS-BusinessActivity. This will be described afterwards.

The coordination protocols described in the WS-AtomicTransaction and WS-BusinessActivity specifications consist of several state machines to choose between. In figure 4 we can see such a state machine, which in this case describes the Two-Phase Commit (2PC) protocol of the WS-AtomicTransaction specification. The main difference between the two specifications is that the Atomic Transaction is meant for short running ACID Transactions, and Business Activity for long running Business Transactions. The other difference is that the Atomic Transaction requires a close trust relation between the resources, where Business Activity does not require this.

**WS-Coordination**

WS-Transaction is a Transaction Management Protocol that makes use of a Coordinator to manage the Transaction. Each service may have its own Coordinator, but a Transaction can also do with only one Coordinator. The Coordinator is an entity that provides the following services, which are thoroughly described in the WS-Coordination specification: [WS-C05].

- Activation Service. Defines a CreateCoordinationContext operation, which may be supported by the Coordinator.
- Registration Service. Allows a web service to register at the Coordinator, this service must be supported by the coordinator.
A set of coordination protocol services. These protocols are defined in the WS-AtomicTransaction and WS-BusinessActivity specifications.

A coordination context is defined at the start of a transaction, by the coordinator of the invoking web service at the request of the web service. Consequently the web service sends the context to the other web service. Which sends it to its own coordinator’s activation service. The registration service of the second coordinator registers at the first registration services with the protocol both applications support. Also the endpoints of the connection are registered and the connection between both web services is created.

**WS-Atomic Transaction**

The real Transactions start when the coordinators have generated the coordination context. The status of the transaction is then active. The coordinator, or coordinators, perform the prepare activity in order to find out if the participants can fulfil the transaction. The participant responds with a positive prepared message or with a negative abort message. If the message is negative, the transaction is rolled back by the coordinator and aborted by both participants. The coordinator responds with a commit upon a prepared message by both participants. The state changes to ended if the participants confirm the coordinator with a committed message.

![Figure 7: Two Phase Commit State Chart [WS_AT].](image)

This 2PC protocol, which can be seen in Figure 7, ensures that all participants in the transaction are mutually aware of the status of the transaction, which makes the transaction predictable and atomic. The protocol can be used for all forms of communication protocols and state machines for the transaction. This holds not only for the communication between participants and coordinators, but also for the communication between the coordinators.

**WS-Business Activity**

The protocols are the most important part of WS-BusinessActivity and also differ from WS-AtomicTransaction. There are two protocols possible:

- BusinessAgreementWithParticipantCompletion
- BusinessAgreementWithCoordinatorCompletion.
The first protocol implies that the participant register with the coordinator for management of the transaction. However the participant is ought to know when its par in the business activity is completed. The second case states that the coordinator has to commit to the participant that the business activity is completed.

Focus of this report is on the protocol with coordinator completion. This is the more complicated form, but has the same states as the participant completion protocol. Since business activities usually take longer than atomic transactions, intermediate states can be so visible to the outside world, in contrast with atomic transactions.

The basic rule in business activities is, that the participants register with the coordinator, while creating the context of the transaction. Continuously, the coordinator requests completion of the transaction by the complete message. Usually the participant completes the transaction and the participant and the coordinator agree on closing and ending the transaction. However in the non-ideal case it might occur that the transactions in the business activity are cancelled, exited of faults occur. The transaction should even then be ended in a legitimate way, by cancelling or exiting. If the transaction has reached the state completed, compensating actions are taken by the participant to get to a valid end state. Faults that occur, always by a participant, result in the state faulting, which makes the coordinator end the business activity with a faulted message and the coordinator knows the participant is out of the transaction. The state chart can be seen in figure 8.

![Figure 8: State chart of Business transaction protocol with coordinator completion[WS-BA]](image)

### 3.2.3 Example

An overview of the example is given in figure 9. It is the example we discussed in section 2.1. The overview is followed by the application of WS-T on the example by creating the coordination context and applying the 2PC protocol. The next section holds the requirements analysis that is based on this example.

The coordinators make the connections between the web services (customer, distributor, carrier and supplier). The web services use these connections to execute the transaction. The 2PC protocol is used on that coordination context between two web services.
Figure 9: Drop Dead Order Example in WS-T with several Coordinators.

The coordinators can only create the coordination context in mutual agreement. The coordination context is necessary in order to arrange communication between the services. The messages are sent in an SOAP envelope over HTTP. The sequence of coordinating the example is given in figure 10. The Sequence starts with the initiating web service, in this case the Customer. It calls its coordinators activation service with createCoordinationContext for a Coordination Context (1). CreateCoordinationContext has to contain the location of the context and may contain a deadline and a current context (e.g. for recovery). Extra attributes and elements might be added for additional information. Next the Coordination Context is sent to the Distributor (2). The Distributor passes the Coordination Context to its coordinators activation service (3), which creates a new coordination context (Cb) with the same transaction id (4). Coordination Context Cb is registered at both registration services and the coordinated connection is made between the Customer and the Distributor (5). The other web services are connected in a similar way (6-10). However they are initiated concurrently and with a new coordination context created by the Distributors coordinator. XML examples of the CreateCoordinationContext message, CreateCoordinationContext response, Register message and Register response can be seen in Appendix A.
Figure 10: Creation of Coordination Context.

**WS-AtomicTransaction**

The transaction can be initiated when the coordination context between at least two web services has been formed. The customer’s coordinator initiates the transaction by sending the prepare message to the participants. In this case the prepare messages are sent to the Customer and the Distributor’s coordinator, which passes the messages to the Distributor. If there is only one coordinator in the transaction, then the coordinator and Distributor are connected directly. The participants respond with the prepared message if they can fulfill the transaction. Otherwise the transaction is aborted, and the other participant is notified. The coordinator commits the transaction upon receiving the positive prepared message. According to the 2PC protocol, the participants have to commit the transaction as well. However the Distributor cannot commit the transaction until it has received a commit from the Carrier and the Supplier. Both will be invoked concurrently in the same pre-described way. If and only if the Distributor has received a commit from the Supplier and Carrier, it can commit the transfer by the coordinator to the Distributor. If the transaction cannot be committed by abortion of the Carrier or Supplier, the entire transaction cannot be fulfilled and the transaction between the Customer and the Supplier had to be rolled back and aborted.
Figure 11: Two Phase Commit Protocol applied on the example.

**WS-BusinessActivity**

The transaction between the Distributor and the Carrier and Supplier illustrate the BusinessAgreementWithCoordinatorCompletion protocol best, thus we will focus on that (Figure 8). The transactions are completed before confirmation is given from the other concurrent translation, which means in case of failure of the other transaction that the transaction has to be compensated. Since the transactions first action is to complete, it is also possible that faults are generated that cannot be compensated or that faults occur during the compensation.

### 3.2.4 Requirements Analysis

1.0 **Atomicity is supported.**

Atomicity is supported through the states of WS-AT and WS-BA, which start with Active and always end with Ended through either completion or some kind of failure. As can be seen in the example, each step of the transaction is reaffirmed by the other participant(s) to ensure an agreement after each step, which eventually leads to an agreed upon commit. Whenever there is not an agreement, the transaction will abort and the coordinator will send the rollback command, which again is reaffirmed with the aborted message by the participants.

1.1 **Rollback is supported.**

Rollback is supported by WS-AT through the rollback message that is included in the WS-AT protocols.
1.2 Compensating actions are supported.
Compensating actions are supported by WS-BA through the compensate message that is included in the WS-BA protocols.

2.0 Consistency is supported.
Consistency is supported through the states of WS-AT and WS-BA (respectively figure 7 and 8), which start with Active and always end with Ended through either completion or some kind of failure. As can be seen in the example, each step of the transaction is reaffirmed by the other participant(s) to ensure an agreement after each step.

2.1 A (composite) Transaction is able to abort.
A (composite) Transaction is able to abort through the Rollback (WS-AT) and Cancel (WS-BA) messages (respectively figure 7 and 8) by the coordinator or through the aborted message by the participant.

2.2 Adding deadlines to Transactions is supported.
Adding deadlines to Transactions is supported by including the Expires attribute in the coordination context XML file. See appendix A1, line 13.

2.3 Logical expressions for specifying constraints are supported.
Logical expressions for specifying constraints are not supported.

3.0 Isolation is supported.
Isolation is supported, and its level can be set with the IsolationLevel attribute in the coordination context. See appendix A1, lines 26-28.

4.0 Durability is supported.
Durability is supported through the states of WS-AT and WS-BA (respectively figure 7 and 8), which require a Committed reaction to the Commit message sent by the coordinator, as can also be seen in the example. Once the Committed message is returned, the transaction is saved by all Participants. However, in the case of the WS-BA protocols, a compensate action might be done after the Committed message.

5.1 Composite Transactions are supported.
Composite Transactions, such as the one in our example are supported. A coordinator allows more then two participants to register for the protocol (or a variant if allowed) and/or allows more then two slave coordinators to register at the master coordinator, like in our example (Figure 9). This allows composite transactions to be supported.

5.2 Distributed Transactions are supported.
Distributed Transactions are supported because the WS-T specification is build upon WSDL and SOAP, which again are built upon XML and HTTP and hence supports Distributed Transactions.

5.3 Transaction recovery by dynamic rebinding and dynamic re-composition at runtime is supported.
Dynamic rebinding and dynamic re-composition is not supported due to it being considered as an implementation issue.

5.4 Secure Transaction of different types can be established. (Confidentiality, Integrity, Authentication and Non-repudiation).
Secure Transaction are not supported by WS-T itself, but by WS-Security.

5.5 Optimistic or pessimistic concurrency control is supported.
Concurrency control is not supported due to it being considered as an implementation issue.
3.3 BPEL


**Abbreviation:** Business Process Execution Language

**Organization:** OASIS

**Language(s):** WSDL, XML schema, XPath, WS-Addressing

### 3.3.1 Overview

BPEL stands for Business Process Execution Language and is built upon WSDL, XML schema, XPath and WS-Addressing. BPEL can be used in for defining business rules and for defining executable business processes (workflows). It defines the business process and calls upon the WSDL interface specification to execute the activity. BPEL can be used in Choreography and Orchestration. The business processes can be either local or distributed and are linked to each other by the partnerlinks. BPEL is supported by e.g. IBM and Microsoft, and is proposed as a standard at OASIS.

### 3.3.2 Details

The business process of a web service is defined in the BPEL file. A BPEL file contains 4 major sections. The partnerlinks holds all the web services the business process interacts with. A partnerlink has four attributes, name, partnerlinktype(specified by WSDL), myRole(optional) and partnerRole(also optional). Variables can be defined in order to send messages to the business partners. Faulthandlers can be defined to catch errors that might occur, such as corrupt data or corrupt communication. Finally there is the sequence of activities. These activities can be found in the BPEL specification, and include for example: invoke, assign, reply and receive. These activities define the real business process.

### 3.3.3 Example

BPEL can be used in Choreography and Orchestration. We implemented the example given in section 2.1 as an Orchestration. Since the example has four web services, there will also be four BPEL files needed. In figure 12, we only give the activity diagram of the distributor process, since it is the most complex one.

The Customer invokes the Distributor with the request for supplying a certain product at a certain location within a given deadline. The distributor invokes on its turn the supplier and the carrier, both with a synchronous request, which means that the transaction stays open until it is completed. Also they are invoked concurrently. The distributor waits on the replies with the receive activity.

Upon receiving both replies, a switch case is executed to decide the distributor's reply. When both replies are positive, the distributor also replies positive and the supplier and carrier are assigned. Otherwise it replies negative and the supplier and carrier are not assigned.
3.3.4 Requirement analysis

1.0 Atomicity is supported.
Atomicity is supported, but has to be forced on implementation level by defining fault- or compensation-handlers. These fault-handlers must perform a roll-back if a failure occurs [BPEL03].

1.1 Rollback is supported
Compensating actions are supported, but have to be forced on implementation level by defining compensation-handlers.

1.2 Compensating actions are supported.
Compensating actions are supported, but have to be forced on implementation level by defining compensation-handlers.

2.0 Consistency is supported.
Consistency can be enforced at the implementation level by fault- or compensation-handlers.

2.1 A (composite) Transaction is able to abort.
A (composite) Transaction is able to abort by the Terminate option.

2.2 Adding deadlines to Transactions is supported.
Adding Deadlines is supported by Deadline- and Duration-valued expressions in XPath 1.0, which result in the XML Schema Date, DateTime and Duration types ([BPEL03] page 40).

2.3 Logical expressions for specifying constraints are supported.
Logical expressions for specifying constraints are supported by Boolean-valued expressions. These Boolean-valued expressions include transition conditions, join conditions, while condition, and switch cases.

3.0 Isolation is supported.
Isolation can only be enforced at the implementation level.
4.0 Durability is supported.
Durability can be enforced at the implementation level.

5.1 Composite Transactions are supported.
Composite Transactions are supported in the form of structured activities. These structured activities include sequence, switch, while, flow and pick statements.

5.2 Distributed Transactions are supported.
Distributed Transactions are supported by definition.

5.3 Transaction recovery by dynamic rebinding and dynamic re-composition at runtime is supported.
Transaction recovery by dynamic rebinding and dynamic re-composition can be implemented at implementation level by use of faulthandlers and/or structured activity functions.

5.4 Secure Transaction of different types can be established.
Secure Transaction are not supported by BPEL4WS itself, but can be added at the implementation level by using WS-Security.

5.5 Optimistic or pessimistic concurrency control is supported.
Concurrency control can be enforced at the implementation level.
3.4 WS-CDL


**Abbreviation:** Web Service Choreography Description Language

**Organization:** Oracle, Commerce One, Novell, Choreology, W3C

**Language(s):** XML

3.4.1 Overview

The Web Services Choreography Description Language (WS-CDL) is an XML-based language that describes cross-enterprise collaborations of Web Services by defining their common observable behaviour. WS-CDL is on the highest level on the current WS-stack, namely on the level of web service composition (figure 1). It is meant to fill the Choreography gap that BPEL leaves open. WS-CDL does support choreographic communication in particular. Currently WS-CDL is up for recommendation at w3c. There is no implementation of the standard available. It is noticeable that this standard, in contrary to the rest of the WS-stack, is not supported by Microsoft and IBM. Also noticeable is the fact that WS-CDL assumes that the business agreements between the participants already have been made.

3.4.2 Details

WS-CDL is used for the description of business processes that interact by choreography. WS-CDL specifies the XML standard for such a choreography package with several optional type definitions. These are used in the choreography notation that defines the actual composed business process. They hold information about how which information is exchanged between the participants.

The type definitions are:

- **informationType**
  Description of the type of information in the choreography to avoid direct referencing to data types.

- **token**
  An alias for a piece of data in a variable or message that needs to be used by a choreography.

- **tokenLocator**
  An XPATH query that selects a token.

- **roleType**
  Specifies the name and functions of the roles the services play in the choreography.

- **relationshipType**
  Specifies the name of the relation and the roleTypes the participants fulfil.

- **participantType**
  Specifies which roleTypes have to be implemented for this participant.

- **channelType**
  Specifies the point of collaboration between participantTypes by specifying where and how information is exchanged. It must specify a roleType and participant reference. It may specify several restrictions for the channel.

- **choreographyNotation**
Defines the choreography of the business activity. Consisting of at least one relationship and activity. Includes optional variable definitions, exceptionblock and a finalizerblock. A choreography package contains at least a root choreography, but a choreography can have several sub choreographies. The exceptionblock is to recover from unusual behaviour by rolling back to the initial state. The finalizerblock is to end the transaction by e.g. a cancel or confirm message. Each different finalising case has its own finalizerblock.

There are several activities that are implementable in a choreography or finalizerblock. They can be structured as sequences, choices and parallel. The activities are e.g. interaction, assign and perform. Exact specifications can be found in [WS-CDL05].

The global structure of a WS-CDL file is as follows:

```xml
<package
    name="NCName"
    author="xsd:string"?
    version="xsd:string"?
    targetNamespace="uri"
    xmlns="http://www.w3.org/2005/10/cdl">
    <informationType/>*
    <token/>*
    <tokenLocator/>*
    <roleType/>*
    <relationshipType/>*
    <participantType/>*
    <channelType/>*

    <choreography name="NCName"
        complete="xsd:boolean XPath-expression"?
        isolation="true"|"false"?
        root="true"|"false"?
        coordination="true"|"false"?
        >
        <relationship type="QName"/>

        variableDefinitions?
        Choreography-Notation*
        Activity-Notation

        <exceptionBlock name="NCName">
            WorkUnit-Notation+
        </exceptionBlock>?

        <finalizerBlock name="NCName">
            Activity-Notation
        </finalizerBlock>*
    </choreography>
</package>
```
3.4.3 Example

In Figure 13 shows in the form of an interaction diagram how WS-CDL can be used to implement the example given in section 2.1. The trigger for this interaction sequence is the placing of the Drop-Dead-Order by the Customer at the Distributor. The Distributor will then check with the Supplier and Carrier if they can comply with the order by a Sub-Choreography, which is nested in the main Choreography. If this Sub-Choreography is finished successful the Distributor will confirm the Transaction to the customer. If one or both of the Sub-Choreographies end with a failure, rollback routines, implemented by the finalizerblocks or exceptionblocks, will be called and the main Choreography will fail.

3.4.4 Requirements Analysis

1.0 Atomicity is supported.
Atomicity is supported, but has to be enforced explicitly by defining proper exceptionBlocks or FinalizerBlocks. These blocks must perform a roll-back if a failure occurs.

1.1 Rollback is supported
Rollback is supported, but must be enforced explicitly by defining exceptionBlocks or FinalizerBlocks.

1.2 Compensating actions are supported.
Compensating actions are supported, but must be enforced explicitly by defining proper exceptionBlocks or FinalizerBlocks.

2.0 Consistency is supported.
Consistency can be enforced at the implementation level by the exceptionBlocks or FinalizerBlocks.

2.1 A (composite) Transaction is able to abort.
A (composite) Transaction is not able to abort unless this is enforced at the implementation level.

2.2 Adding deadlines to Transactions is supported.
Adding deadlines is supported by using the timeout element of the interaction-constructs within a choreography construct[WS-CDL05, section 5.4].

2.3 Logical expressions for specifying constraints are supported.
Logical expressions for specifying constraints are supported. Predicate expressions [WS-CDL05, section 5.3] can be used within WS-CDL to specify conditions. Query expressions[WS-CDL05, section 5.3] are used within WS-CDL to specify queries. The language used in WS-CDL for specifying expressions and queries or conditional predicates is XPath 1.0.

3.0 Isolation is supported.
Isolation is supported by the optional isolation attribute. This isolation attribute specifies whether sibling Choreographies can access the variables within the Choreography.

4.0 Durability is supported.
Durability has to be enforced at the implementation level.

5.1 Composite Transactions are supported.
Composite transactions are supported because a choreography-notation is recursive.

5.2 Distributed Transactions are supported.
Distributed transactions are supported due to the distributed nature of the invoked web services.

5.3 Transaction recovery by dynamic rebinding and dynamic re-composition at runtime is supported.
Dynamic recovery of a transaction is not supported because the business agreement, on which the choreography is based, has to be defined at design time.

5.4 Secure Transaction of different types can be established.
Security measures can be implemented by adhering to the WS-Security specification.

5.5 Optimistic or pessimistic concurrency control is supported.
Concurrency is optional, by a parallel activity structure. It depends on the implementation of the Choreography (of third parties) if it is optimistic or pessimistic.
4 Evaluation

In the past two chapters we have discussed four standards regarding Web Service Transactions and Compositions. We evaluated each of them by going through the requirements listed in chapter 2 and elaborated if the standard supported it, and why it did or did not. To summarize we have listed the conclusions of those evaluations into table 3.

<table>
<thead>
<tr>
<th>(Failure) Atomicity</th>
<th>BTP</th>
<th>WS-AT</th>
<th>WS-BA</th>
<th>BPEL</th>
<th>WS-CDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomicity is supported.</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Rollback is supported.</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Compensating actions are supported.</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Consistency</th>
<th>BTP</th>
<th>WS-AT</th>
<th>WS-BA</th>
<th>BPEL</th>
<th>WS-CDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consistency is supported.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>A (composite) Transaction is able to abort.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Adding deadlines to Transactions is supported.</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Logical expressions for specifying constraints are supported.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Isolation</th>
<th>BTP</th>
<th>WS-AT</th>
<th>WS-BA</th>
<th>BPEL</th>
<th>WS-CDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isolation is supported.</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Durability</th>
<th>BTP</th>
<th>WS-AT</th>
<th>WS-BA</th>
<th>BPEL</th>
<th>WS-CDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Durability is supported.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Miscellaneous</th>
<th>BTP</th>
<th>WS-AT</th>
<th>WS-BA</th>
<th>BPEL</th>
<th>WS-CDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composite Transactions are supported.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Distributed Transactions are supported.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Transaction recovery by dynamic rebinding and dynamic re-composition is supported.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Secure Transaction of different types can be established. (Confidentiality, Integrity, Authentication and Non-repudiation)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Optimistic or pessimistic concurrency control is supported.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Supports long running Transactions.</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Specification Language</th>
<th>XML</th>
<th>WSDL &amp; XSD</th>
<th>WSDL &amp; XSD</th>
<th>BPEL</th>
<th>XML</th>
</tr>
</thead>
</table>

Table 3: Evaluation matrix.

X     Supported
-     Not supported

Looking at table 2, we notice several things.

• First we see that WS-T actually consists of two Transaction Protocols, namely WS-AT and WS-BA. Together with the WS-Coordination framework, this could make it hard to comprehend for someone who is not familiar with the subject. The big advantages of WS-T are the fact that it is part of the standard W3C WS-Stack and its support of long running Transactions. Long running Transactions are specifically supported by
WS-BA, WS-AT is impractical for these Transactions. Extensive compensation actions may be needed with WS-BA, since it relies on compensating actions completely.

- BTP at the other hand is much easier to comprehend because it is specified in much more detail. Another difference is that BTP has included confirmsets. These confirmsets let the Application Element choose which parties in the Transaction are to be cancelled and which are to be confirmed. In this way, the Application Element is able to contact more services which perform the same task and to choose the best option the big disadvantage of BTP is that it is not part of the WS-Stack. BTP also does not support long running Transactions.

- There is also a difference in granularity between these Transaction standards. WS-AT contains simple 2PC protocols, WS-BA contains non 2PC non blocking protocols and BTP his protocol consists of a sequence of small atomic Transactions.

- WS-Security can be combined with WS-T as well as with BTP, to take care of security issues.

- We also notice that BPEL is the only standard that supports dynamic rebinding and composition, but only at the implementation level. BPEL has become the standard language in the field and knows many implementations, including open source implementations.

- WS-CDL is specifically designed for Choreography, while BPEL supports both Orchestration and Choreography. However, according to [DUST06] some say the Choreography support of BPEL is not ideal.

- The big disadvantage of WS-CDL is that it is not supported by the large players in the field and that does not have any implementations yet.

We recommend using WS-T for Transactions and BPEL for the implementation. WS-T is preferred since it supports long running Transactions and is part of the WS-Stack, and BPEL is preferred because of its support of the large companies and can be used for both Orchestration and Choreography. We also recommend adding confirmsets as used by BTP to WS-T in its next incarnation. This feature is most useful in Transactions and would be a welcomed addition to WS-T because it then is defined by the specification that one can select the service with the highest QoS. Adding long running Transactions to BTP is also recommended. This could be done by structuring logic long-lived business transactions into small atomic transactions and using cohesion to connect these atomic transactions (address scope and blocking).
References

[BTP02] OASIS, Business Transaction Protocol 1.0, 2002
[BPEL02] IBM/Microsoft ea, BPEL for WS, 2003
[DUST06] Prof. Dr. S. Dustdar, Lecture notes information systems, 2006
[RFC2119] S. Bradner, Key words for use in RFCs to Indicate Requirement Levels, 1997
[MATJ] M. Juric, A Hands-on Introduction to BPEL
[WS-C05] Cabrera ea., Web Services Coordination, 2005
[WS-CDL05] W3C, Web Services Choreography Description Language Version 1.0, 2005
## Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ACID properties</strong></td>
<td>Key features given to Transactions, where ACID stands for Atomicity, Consistency, Isolation, and Durability.</td>
</tr>
<tr>
<td><strong>Actor</strong></td>
<td>A role of an entity external to the system.</td>
</tr>
<tr>
<td><strong>Atomic Transaction</strong></td>
<td>A Transaction that either finishes completely, or not at all.</td>
</tr>
<tr>
<td><strong>Atomicity</strong></td>
<td>The ability of a transaction that it either finishes completely, or not at all</td>
</tr>
<tr>
<td><strong>Authentication</strong></td>
<td>Verification of the identity of a user or the user's eligibility to access an object.</td>
</tr>
<tr>
<td><strong>Binding</strong></td>
<td>The creation of a simple reference to something which is larger and more complicated and used frequently. The simple reference can be used instead of having to repeat the larger thing.</td>
</tr>
<tr>
<td><strong>Business Process</strong></td>
<td>The sequence of process steps a business makes in response of an event.</td>
</tr>
<tr>
<td><strong>Business process Description</strong></td>
<td>Description of/a way of describing a Business Process.</td>
</tr>
<tr>
<td><strong>Business Transaction</strong></td>
<td>An economic event that changes the financial position of an organization; often takes the form of an exchange of economic consideration (such as goods, services, money, or rights to collect money) between two parties.</td>
</tr>
<tr>
<td><strong>Choreography</strong></td>
<td>A model encompassing all parties and their associated Interactions. Antonym to orchestration.</td>
</tr>
<tr>
<td><strong>Collaboration</strong></td>
<td>Business Transaction</td>
</tr>
<tr>
<td><strong>Compensating Actions</strong></td>
<td>Actions to undo state changes in the event of a transaction failure.</td>
</tr>
<tr>
<td><strong>Composition</strong></td>
<td>A set of bindings that form a Transaction.</td>
</tr>
<tr>
<td><strong>Composite Transaction</strong></td>
<td>A (Nested) Transaction between more then two partners.</td>
</tr>
<tr>
<td><strong>Confidentiality</strong></td>
<td>Prevention of disclosure, to other than authorized individuals.</td>
</tr>
<tr>
<td><strong>Consistency</strong></td>
<td>The ability of a transaction to begin and end in a legal state and thus it not breaking any integrity constraints.</td>
</tr>
<tr>
<td><strong>Conversation</strong></td>
<td>An exchange of (Transaction) messages between parties.</td>
</tr>
<tr>
<td><strong>Distributed Transaction</strong></td>
<td>A Transaction between two or more network hosts.</td>
</tr>
<tr>
<td><strong>Durability</strong></td>
<td>The ability of a transaction to persist after one has been notified on success, and cannot be undone.</td>
</tr>
<tr>
<td><strong>Dynamic Rebinding</strong></td>
<td>The possibility of forming a new binding at runtime with a</td>
</tr>
</tbody>
</table>
different participant when the current participant is not able to fulfil its promises. Whereas the new participant has to perform the same functionality as the failing participant.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Re-composition</td>
<td>The runtime forming a new composition by replacing one binding with a composition or by replacing a sub composition by a binding. In case of failure of a sub composition or binding.</td>
</tr>
<tr>
<td>Dynamic Transactions</td>
<td>A transaction that can be change at runtime by either dynamic re-composition or dynamic rebinding.</td>
</tr>
<tr>
<td>Heterogeneous Environment</td>
<td>An environment consisting of diverse or dissimilar parts; having non-uniform structure or composition.</td>
</tr>
<tr>
<td>Integrity</td>
<td>Means the property that data or information have not been altered or destroyed in an unauthorized manner.</td>
</tr>
<tr>
<td>Interoperability</td>
<td>The ability of several web services to operate together, with interfaces to overcome problems in communication.</td>
</tr>
<tr>
<td>Isolation</td>
<td>The ability of a transaction to make operations isolated from all other operations.</td>
</tr>
<tr>
<td>Messaging Framework</td>
<td>Specification of guidelines and restrictions that have to be adhered to improve the communication.</td>
</tr>
<tr>
<td>Modelling Methodology</td>
<td>Methodology, consisting of methods and techniques, to develop and visualise models of compositions.</td>
</tr>
<tr>
<td>Nested Transaction</td>
<td>A Transaction consisting of multiple smaller transactions.</td>
</tr>
<tr>
<td>Non-repudiation</td>
<td>The condition whereby the sender of a message cannot deny the validity of the result of the process used to authenticate the data.</td>
</tr>
<tr>
<td>Orchestration</td>
<td>The organisation of a composition, the opposite of a choreographic organisation. Orchestration languages have a central coordinating process, the transaction manager, that assigns the other services.</td>
</tr>
<tr>
<td>Participant</td>
<td>A business or other organisation providing a web service that is accessible by other participants by a (pre) defined binding.</td>
</tr>
<tr>
<td>Quality of Service</td>
<td>A concept with which clients can indicate the level of service they require.</td>
</tr>
<tr>
<td>Transaction</td>
<td>Mechanism for ensuring that all participants in an application achieve a mutually agreed outcome.</td>
</tr>
<tr>
<td>Transaction Management</td>
<td>A class of standards that do not implement a transaction or model a composition, but specify how a transaction should be implemented and check that participants behave according to the predefined way.</td>
</tr>
<tr>
<td>Transaction Pattern</td>
<td>The pattern that describes the way the transaction behaves.</td>
</tr>
<tr>
<td>Transport Protocol</td>
<td>A transport protocol describes how the messages are sent.</td>
</tr>
</tbody>
</table>
to the participants, for example by SOAP.
| Web Service | A service made available by an organisation, presumably a business, over the web. Usually a web service applies to the standards of UN/CEFACT, OASIS or W3C. It has to be interoperable, usually by means of an interface. Also a web service itself is considered a black box on the technical level. |
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACID</td>
<td>Atomicity, Consistency, Isolation and Durability</td>
</tr>
<tr>
<td>B2B</td>
<td>Business to Business</td>
</tr>
<tr>
<td>BPEL</td>
<td>Business Process Execution Language</td>
</tr>
<tr>
<td>BTP</td>
<td>Business Transaction Protocol</td>
</tr>
<tr>
<td>ebXML</td>
<td>electronic business eXtensible Mark-up Language</td>
</tr>
<tr>
<td>EDI</td>
<td>Electronic Data Interchange</td>
</tr>
<tr>
<td>OASIS</td>
<td>Organization for the Advancement of Structured Information Standards</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Definition Language</td>
</tr>
<tr>
<td>WS-*</td>
<td>The collective name for the WS specification framework</td>
</tr>
<tr>
<td>WS-T</td>
<td>Web Service - Transactions</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Mark-up Language</td>
</tr>
</tbody>
</table>
Appendix

Appendix A: WS-Coordination XML Messages.

Appendix A1: Create CoordinationContext [WS-C05]

```xml
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Header>
    ... 
    <wscoor:CoordinationContext
      xmlns:wscoor="http://schemas.xmlsoap.org/ws/2004/10/wscoor"
      xmlns:myApp="http://fabrikam123.com/myApp"
      S:mustUnderstand="true">
      <wscoor:Identifier>
        http://Fabrikam123.com/SS/1234
      </wscoor:Identifier>
      <wscoor:Expires>3000</wscoor:Expires>
      <wscoor:CoordinationType>
        http://schemas.xmlsoap.org/ws/2004/10/wsat
      </wscoor:CoordinationType>
      <wscoor:RegistrationService>
        <wsa:Address>
          http://.../registration
        </wsa:Address>
        <wsa:ReferenceProperties>
          <myApp:BetaMark> ... </myApp:BetaMark>
          <myApp:EBDCode> ... </myApp:EBDCode>
        </wsa:ReferenceProperties>
      </wscoor:RegistrationService>
      <myApp:IsolationLevel>
        RepeatableRead
      </myApp:IsolationLevel>
    </wscoor:CoordinationContext>
    ... 
    <S:Body> 
    ... 
  </S:Body>
</S:Envelope>
```
Appendix A2: Create CoordinationContext Response [WS-C05]

```xml
<CreateCoordinationContextResponse>
  <CoordinationContext>
    <Identifier>
      http://Business456.com/tm/context1234
    </Identifier>
    <CoordinationType>
      http://schemas.xmlsoap.org/ws/2004/10/wsat
    </CoordinationType>
    <RegistrationService>
      <wsa:Address>
        http://Business456.com/tm/registration
      </wsa:Address>
      <wsa:ReferenceProperties>
        <myapp:PrivateInstance>
          1234
        </myapp:PrivateInstance>
      </wsa:ReferenceProperties>
    </RegistrationService>
  </CoordinationContext>
</CreateCoordinationContextResponse>
```

Appendix A3: Register [WS-C05]

```xml
<Register>
  <ProtocolIdentifier>
    http://schemas.xmlsoap.org/ws/2004/10/wsat/Volatile2PC
  </ProtocolIdentifier>
  <ParticipantProtocolService>
    <wsa:Address>
      http://Adventure456.com/participant2PCservice
    </wsa:Address>
    <wsa:ReferenceProperties>
      <BetaMark> AlphaBetaGamma </BetaMark>
    </wsa:ReferenceProperties>
  </ParticipantProtocolService>
</Register>
```

Appendix A4: Register Response[WS-C05]

```xml
<RegisterResponse>
  <CoordinatorProtocolService>
    <wsa:Address>
      http://Business456.com/mycoordinationservice/coordinator
    </wsa:Address>
    <wsa:ReferenceProperties>
      <myapp:MarkKey> %%F03CA2B%% </myapp:MarkKey>
    </wsa:ReferenceProperties>
  </CoordinatorProtocolService>
</RegisterResponse>
```
Appendix B: Main BTP State Charts

Appendix B1: Superior’s State Chart [BTP02]
Appendix B2: Inferior’s State Chart [BTP02]