Use Cases for the Business Transaction Protocol

OASIS Business Transactions Technical Committee
Models/Use Cases Technical Subcommittee
bt-models@lists.oasis-open.org

Introduction

This document attempts to capture the diversity of transactional requirements that may be imposed by applications which are built around Web Services. The transactional models implied by the presented scenarios are not the only ones that may be devised. I must be mentioned that the Business Transaction Protocol (BTP) will not support all possible transactional models in its first incarnation [reference to the model document].

Web Services

The arena that BTP mainly targets is that of Web Services where applications, which may span enterprise boundaries, are built through the composition of loosely-coupled components using protocols like SOAP, WSDL, UDDI, etc., as the gluing mechanism. BTP allows distinct Web Services to participate within the context of one business transaction. In the scenarios that follow, every component is considered to be a Web Service, even when not explicitly identified as such.

Use Cases

1. Arranging a Night-Out
2. Home entertainment system
3. Arranging a meeting
4. Manufacturer-Supplier-Shipper
5. Financial Trading
6. Micro-payment

Status of this Document

Revision 0.5 – Last updated 03/05/2001 14:01

Editor

Savas Parastatidis
HP Arjuna Labs
Savas.Parastatidis@arjuna.com

1. Arranging a Night-Out

Consider an individual wishing to arrange a night-out in a city where all business services are also available as Web Services. The individual concerned uses a Personal Digital Organiser (PDA) to control the activities required for making the necessary arrangements with different service providers (e.g., "Taxi," "Theatre," "Restaurant," and "Hotel").

The different activities are to be enclosed within the borders of a single long-running business transaction to achieve an all-or-nothing result and to accommodate for failures. According to BTP terminology, the PDA is acting as the "Initiator" of the business transaction. It also acts as a "Terminator" since it is the source of the decision on the outcome of the business transaction.

The Web Services must be "BTP-aware" in order to participate in the business transaction. They may be already known to the Initiator or they may be dynamically located through UDDI. In any case, Figure 1 illustrates the business transaction that is required for arranging the night-out. The application activity is concerned with booking a taxi (t1), reserving a table at a restaurant (t2), reserving a seat at the theatre (t3), and then booking a room at a hotel (t4). The four activities involve the "Taxi," "Theatre," "Restaurant," and "Hotel" Web Services respectively. If all of these operations were performed as a single transaction (shown by the dotted ellipse), then resources acquired during t1 would not be released until the top-level transaction has terminated. If subsequent activities t2, t3 etc. do not require those resources, then they will be needlessly unavailable to other clients.

Long-running applications and activities can be structured as many independent, short-duration top-level transactions, to form a long-running business transaction. This structuring allows an activity to acquire and use resources for only the required duration of this long-running transactional activity. Therefore, as shown the business transaction may be structured as many different, coordinated, short-duration top-level transactions.

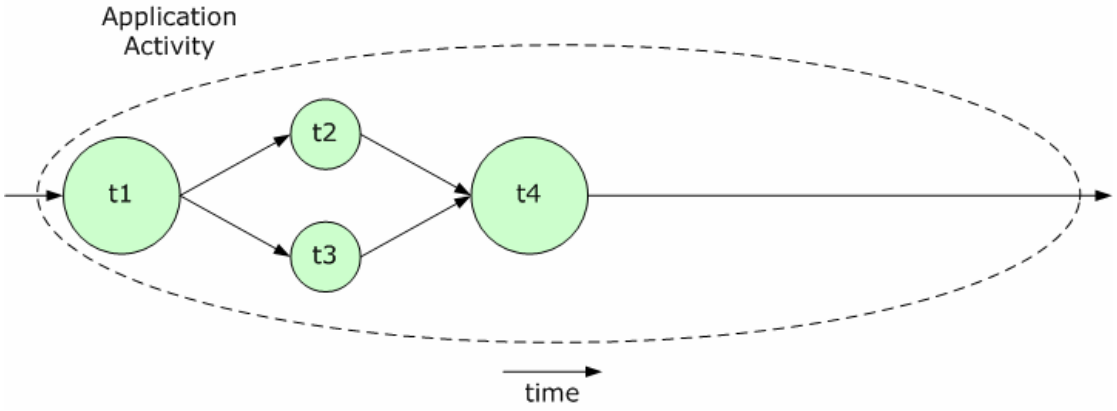


Figure 1: An example of a logical long-running "transaction", without failure.

However, if failures and concurrent access occur during the lifetime of these individual transactional activities then the behaviour of the entire "logical long-running transaction" may not possess ACID properties. Therefore, some form of (application specific) compensation may be required to attempt to return the state of the system to (application specific) consistency. Just as the application programmer has to implement the transactional work in the non-failure case, so too will programmers typically have to implement compensation transactions, since only they have the necessary application

specific knowledge. Note, for simple or well-ordered work it is possible to provide automatic compensations.

For example, let us assume that t4, the activity of reserving a room through the "Hotel" Web Service, has failed (rolls back). Further assume that the application can continue to make forward progress, but in order to do so must now undo some state changes made prior to the start of t4 (by t1, t2 or t3). For example, the individual arranging the night-out decides that since it was not possible to reserve a room at a hotel, the plans will have to change for the night and instead of going to the theatre and the restaurant (t2 and t3 respectively), the cinema and then eating at home will be preferable. The taxi would still be required though. Therefore, new activities are started: tc1 is a compensation activity that will attempt to undo the state changes performed by t2 and t3; t5 and t6 are new activities that continue after the compensation (the "Cinema" and "Food Delivery" Web Services are involved respectively). Obviously other forms of transaction composition are possible.

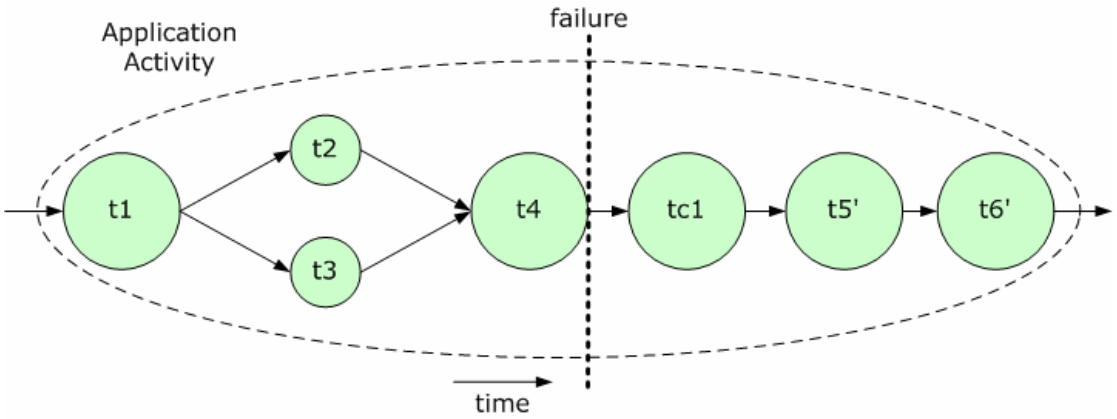


Figure 2: An example of a logical long-running "transaction", with failure.

It should be noted that even with suitable compensations, it can never be guaranteed to make the entire activity transactional: in the time between the original transaction completing and its compensation running, other activities may have performed work based upon the results of the yet to be compensated transaction. Attempting to undo these additional transactions (if this is possible) can result in an avalanche of compensations that may still not be able to return the system to the state it had prior to the execution of the first transaction. In addition, compensations may (continually) fail and it will then be extremely important to inform users (or system administrators). Note, it will be application specific as to whether or not a compensation should be tried again if it does fail. For example, consider the situation where a transaction sells shares and the compensation is to buy them back; if the compensation fails it may be inappropriate (and expensive) to try it again until it does eventually succeed if the share price is going up rapidly.

2. Home entertainment system

Let us assume that we are interested in building our own customised home entertainment system consisting of TV, DVD player, hi-fi and video recorder. Furthermore, rather than purchase each of these from the same manufacturer we want to shop around and get the best of each from possibly different sources. In order to do so, we want to contact different Web Services offering online purchase of home entertainment system parts.

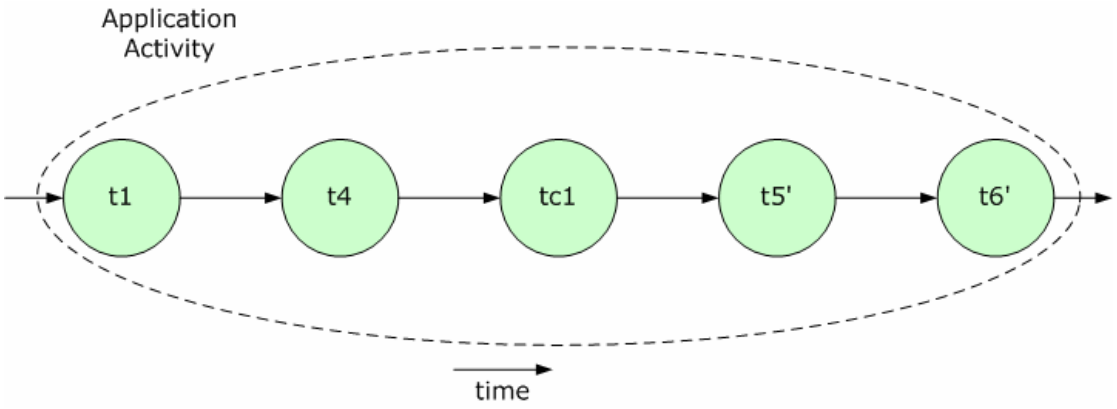


Figure 3: Building a home entertainment system via the Web.

When we contact the "TV" Web Service we wish to start a transactional activity, t1, that will allow us to search and provisionally reserve (obtain transactional locks on) a number of different televisions (set A) that match our requirements. Before t1 terminates, we select a subset of the televisions, B, we are interested in, and provide a reference to our online bank account which the television site may contact to check that we have sufficient funds. t1 then ends, any locks obtained that are not members of B are released as normally for a transaction, (allowing other users to acquire them immediately if necessary), and all other locks are obtained and passed to t2.

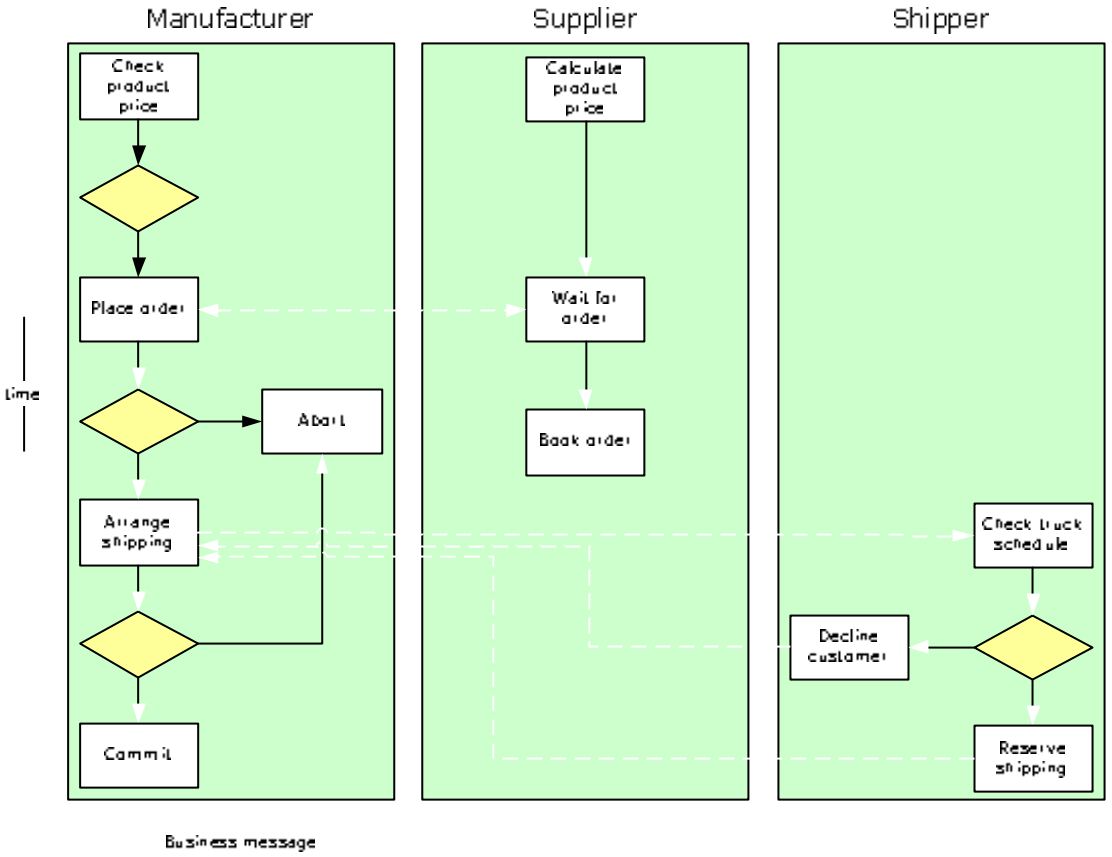
The sequence of operations for t2, t3, and t4 are identical, where only subsets of items (DVD player, hi-fi, video) we have transactionally locked are released when each activity terminates. By the time t5 is executed there is a list of items that are locked and under its control, possibly also including the on-line bank account. Therefore, t5 is responsible for committing or rolling back the final purchase order. All locked resources are atomically handed off to the next atomic action, and failures do not require compensation: the atomicity property of t1, t2, t3, t4 and t5 ensures that either the purchase happens, or it does not (and all resources will be released).

3. Arranging a meeting

The requirement is to arrange a date for a meeting between groups of people; it is assumed that each user has a personal diary, which is exposed as a Web Service, and that each diary entry (slot) can be locked separately. The application starts by informing people of a forthcoming meeting and then receiving from each a set of preferred dates. Once this information has been gathered, it will be analyzed to find the set of acceptable dates for the meeting. This set is then broadcast to the users to get a more definitive idea of the preferred date(s). This process is repeated until a single date is determined. To reduce the amount of work which must be re-performed in the event of failures, and to increase the concurrency within the application, it would be desirable to execute each "round" of this protocol as a separate top-level transaction. However, to prevent concurrent arrangement activities from conflicting with each other, it would be beneficial to allow locks acquired on preferred diary entries to be passed from one transaction to another, i.e., the locks remain acquired on only those entries which are required for the next "round".

4. Manufacturer-Supplier-Shipper

Let's consider an example business transaction scenario depicted in Figure 4, where the "Manufacturer," the "Supplier," and the "Shipper" are all Web Services.


Manufacturer
Supplier
Shipper
Check product price
Calculate product price
Place order
Wait for order
Time
Abort
Book order
Arrange shipping
Check truck schedule
Decline customer
Commit
Reserve shipping
Business message

- *Consistent*: If the parts get ordered, the shipping gets set up. If the shipping company cannot promise shipping with the required terms, the order is cancelled.
- *Durable*: All parties persist the outcome of the transaction.

Typically, the Supplier will have multiple business transactions with multiple Manufacturers executing concurrently. The time between the order is placed and it is confirmed can be long. Therefore it is not feasible for the Supplier to lock its order database and wait until the confirmation comes. It will rather book the order when it is placed and in case the order gets cancelled, it will invoke a compensating action to remove the order from the database. This means that concurrent business transactions are not executing in isolation: they are exposed to partial updates made by other concurrently executing transactions.

These requirements are not special to this example; in fact they must be met for any transaction that is critical to the business of the participating companies. Therefore an application independent facility should exist that can manage the mission critical multi-company business transactions to ensure the properties above.

5. Financial Trading

Many financial trading strategies are composed of several underlying financial products.

For example, an investor wishes to buy stock in company A in the belief that the stock price will rise, but at the same time wishes to 'insure' the investment against an unexpected drop in the stock price.

This strategy can be achieved by buying the stock in company A and at the same time paying a premium to purchase an option to sell the stock at an agreed price (a 'put' option). If the stock price rises the return for the investor is the price gain less the premium paid for the option. If the price falls unexpectedly, the investor exercises the option to sell the stock and limits the loss.

To construct this trade the investor requests quotes on the current price of A's stock and also on the 'put' option on stock A. Quotes on the stock and the option may be available directly from several exchanges, or indirectly through intermediaries or brokers. Prices may vary from exchange to exchange and from broker to broker for many reasons, including differing transaction costs, broker spreads and option valuation models.

The 'value' of such a trading strategy requires the successful purchase of each component or 'leg' of the trade at a certain price and at the same time.

Using a cohesion protocol the investor can automate the quote gathering, quote choosing and execution of the trade.

Within a single cohesion, the investor gathers quotes for the stock and the option from a variety of exchanges and brokers. The quotes are only valid for a specified time after which they 'time-out'. The stock quotes are sorted according to some defined business logic and then the cheapest executed. Similarly, the option quotes are sorted according to some defined business logic and then the cheapest executed. The other quotes are ignored and simply time-out.

6. Micro-payment

Given the size of its potential customer base, the Internet is the first communications media which can reasonably support a micro-payment based economic environment. The notion of micro-payments are that one pays very little, in fact next-to-nothing in

keeping with the notion that everything on the web is (nearly) free, for a product or service on the Internet, and that value is extracted from vast numbers of such small payments being transacted.

The most appropriate commercial scenarios that could make use of micro-payments are those which can be conducted entirely on-line and do not involve shipping of physical, real-world items which mandates (in micro-payment terms) a significant cost and is therefore prohibitive. The distribution of multimedia and software are likely candidates for micro-payment, as are other information services such as stock quotes, weather reports, and suchlike.

In this use case, we shall examine a potentially commonplace scenario, where electronic media (such as MP3 files) can be bought over the Internet for a few pennies. In this case we require a media service and a micro-payment service as our basic application components.

Initially, the client application contacts the media service and locates the file that the user is interested in purchasing. Once located, the pricing details of that file are returned to the client application which then proceeds to authenticate with the micro-payment service. In transactional terms, these interactions would also have the effect of enlisting the services with the transaction manger. This is illustrated in Figure 5:

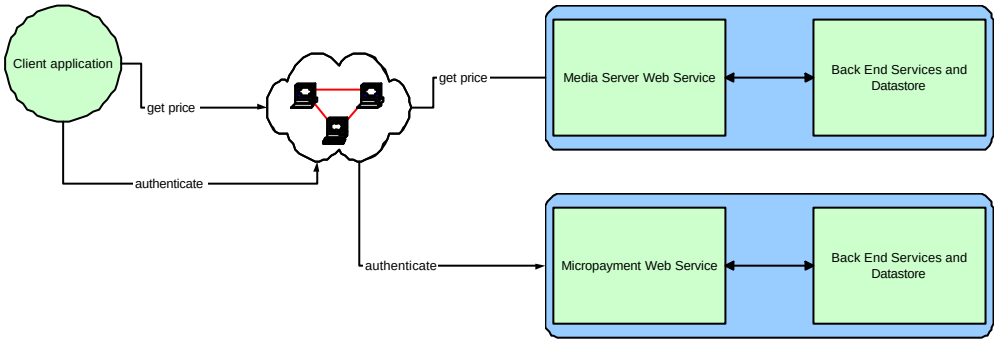


Figure 5: The Initial Stages of the Application (The "Enlistment" Phase)

Once the client application has been authenticated with the micro-payment service, it can request a token from that service which can be used to locate the desired amount of cash at some point in the future (for security purposes this token is not in "activated" at this point). Once the client application receives this token, it can use it to buy the media from the media service by passing the token obtained from the micro-payment service. In return for the micro-payment token, the client application receives a media token from the media service. As with the micro-payment token, the media token is inactive at the moment of issue and will remain that way until the media service decides to make it active. In transaction terms, this would regarded as being the prepare phase. This is shown in Figure 6 below.

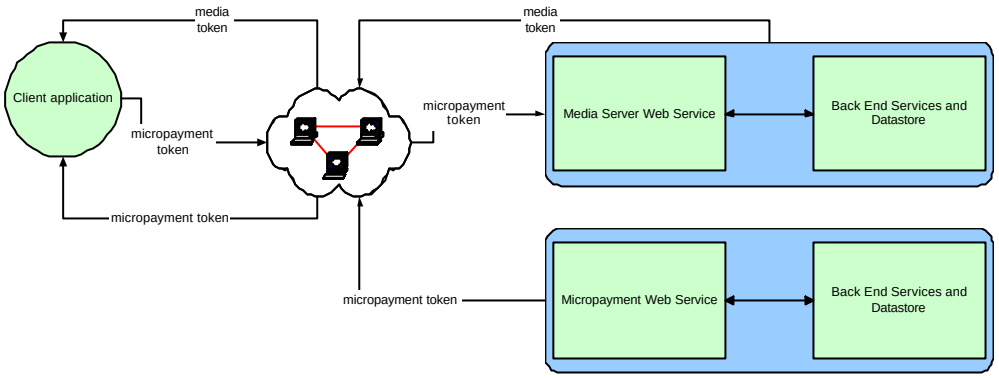


Figure 6 Exchange of Payment Tokens Amongst Web Services (The "Prepare" Phase)

The final stage of this use case is to actually activate both the media and payment tokens such that the media service can draw an actual payment from the micro-payment service and the client application can download a file from the media service. This can be seen as being analogous to the commit phase in transaction terms. Once the parties are satisfied that they have all "committed" then both the media and micro-payment Web Services will activate their respective tokens. The client application is then able to ask for a receipt from the micro-payment service for the payment token used, and may now use its recently activated media token to download the required file from the media service. Behind the scenes, the media service must present its payment token to the micro-payment service, but this is of little interest to the client application and can be largely ignored for the purposes of this use case. The final step can be seen in Figure 7. Note that in Figure 7 it is implied that some entity has verified that both Web Services are able to proceed and has in effect "committed" them.

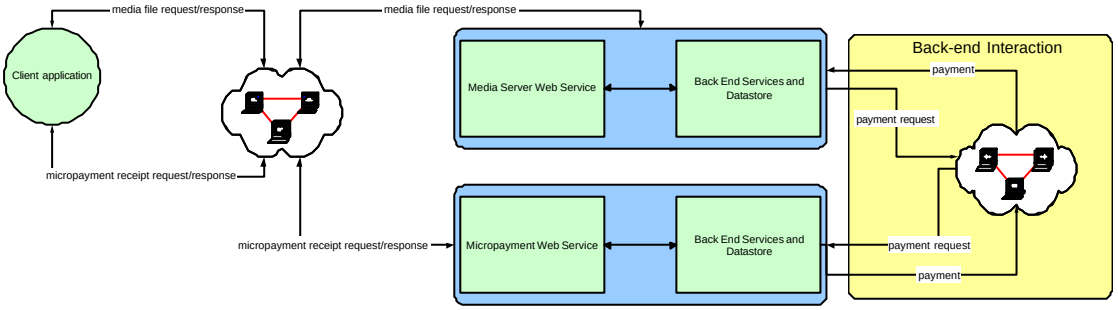


Figure 7 Issuing of Receipt and Media File (The "Commit" Phase)

This is a very ordinary use of transactioning and does not solve (nor seek to solve) any of the usual failure modes such as servers crashing and networks becoming unavailable. In particular it assumes that the Web Services are both trustworthy and will not collude to swindle the client application.

The other important difference to current more heavyweight uses of transactions is the issue of cost. Since micro-payments are inherently small, the cost of transacting such a payment must also be small. Since real value can only be achieved through large volumes of such transactions, the supporting architecture must be performant and scalable.