Proposal for
Business Transaction Protocol
Version 1.0

Sanjay Dalal (sanjay.dalal@bea.com)
Pal Takacsi-Nagy (pal.takacsi@bea.com)

Abstract

Long lasting business transactions spanning multiple enterprises pose a unique challenge to B2B systems. The interdependent workflows among multiple trading partners, which drive business transactions, need to be coordinated to ensure that the outcome of the transaction is reliable. In this document we propose a solution to this problem in the form of a *Business Transaction Protocol (BTP).* B2B servers participating in business transactions over the Internet are expected to implement BTP to orchestrate multi-enterprise transactions.

Status of this Document

This document is a proposal from BEA Systems, Inc. to the OASIS Business Transactions Protocol Technical Committee. The BTP TC has not approved it.

Distribution of this first draft document is limited to members of the BTP TC.

Contents

Introduction

In this document we propose a protocol, called Business Transaction Protocol, which can be used to orchestrate long running, inter-enterprise business transactions. This protocol addresses the unique requirements of business-to-business transactions. BTP is based on the multi-level transaction model that provides the necessary independence for the participating resource managers – in this case the B2B servers of companies engaging in business transactions.
This document is not a complete specification; it is rather a proposal that is intended to serve as a starting point for the work of the OASIS Technical Committee on Business Transactions.

Approach

What is a business transaction

A *business transaction* is a consistent change in the state of the business that is driven by a well-defined business function. Business transactions can be fully or partially automated. Business processes are composed of several business transactions. An example of a business transaction is an order. The function is well defined: order some goods from a company. The completion of an order results in a consistent change in the state of the affected business: the order database is updated and a paper copy of the purchase order is filed.

What is a business-to-business transaction

Ordinary business transactions are usually an interaction between a person and a company. *Business-to-Business (B2B) transactions* on the other hand – as the name suggests – are interactions between businesses. B2B transactions are business transactions that are more often automated and are usually more complex than ordinary business transactions. In the following we summarize the characteristics of B2B transactions:

- They represent a function that is critical to the business, such as supply-chain management
- They are long running
- They can involve more than two parties (companies) and multiple resources operated independently by each party, such as mainframe applications and ERP systems
- They utilize machines-to-machine communication
- They are based on a formal trading partner agreement, like RosettaNet PIPs or ebXML Collaboration Protocol Agreements

Enabling automated B2B transactions is a substantial undertaking, which involves all aspects and functions of middleware, such as messaging, transaction and workflow management, authorization, data security, etc.

In this paper we focus on the transaction management for B2B transactions.

Transaction Management

A typical automated transaction involves multiple resources such as databases, logs, transactional objects and persistent queues. The goal of transaction management or transaction coordination is to orchestrate the termination that can be either committing or rolling back all updates to all resources of the transaction. This improves reliability, manageability and accountability of systems utilizing transaction management.

Example Scenario

Let's consider an example business transaction scenario depicted in Figure 1.

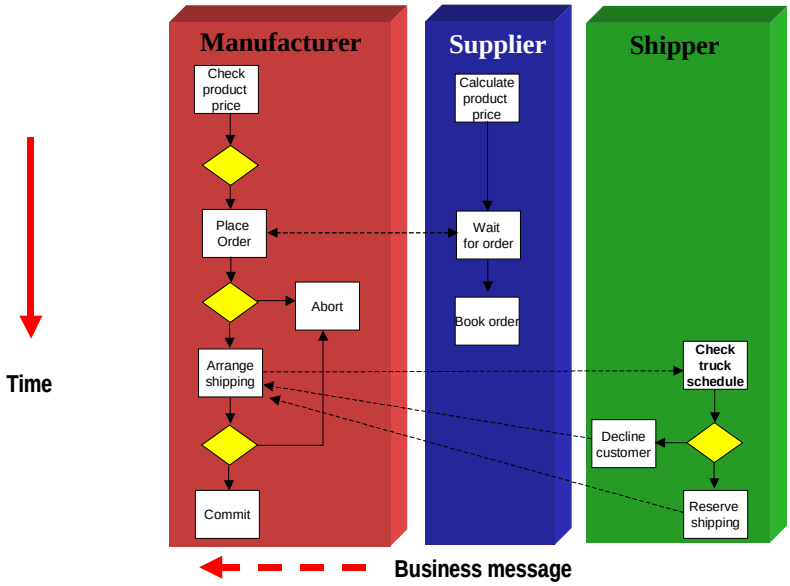

Figure 1 Example scenario

A manufacturing company (*Manufacturer*) needs to order parts from one of its partners (*Supplier*). In order to make its production schedule, the Manufacturer has to make sure that the parts are shipped from the Supplier in a given timeframe by a logistics provider (*Shipper*), otherwise the Manufacturer would not be interested in these parts. All the parties in this example have automated computer systems that can communicate with each other via XML messages. Below we describe the interactions of this business transaction:

1. *Manufacturer's* production scheduling system sends an Order message to *Supplier*
2. *Supplier's* order processing system sends back an order confirmation with the details of the order

3. *Manufacturer* orders delivery from *Shipper* for the ordered parts. Delivery needs to occur in two days.
4. *Shipper* evaluates the request and based on its truck schedule it sends back a confirmation or a "can't do" message.
5. *Manufacturer* either confirms the order and the shipping or it cancels the order, since the shipper was unable to fulfill the request

From the Manufacturer's point of view all the business messages described above belong to a single business transaction. The underlying systems need to make sure that this business transaction is

- *Atomic*: The parts either get ordered or the order gets cancelled
- *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 transaction 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.

Transaction models

In this section we define a transaction model for business-to-business transactions (see [3] for a detailed discussion on transaction terminology and models).

The most straightforward and well-known transaction model is the "flat transaction model". Flat transactions possess the well-known ACID properties: atomicity, consistency, isolation and durability. This model provides a single layer of completion control. One cannot commit or abort parts of a flat transaction or commit results in steps. Isolation is achieved by locking the resources involved for the duration of the transaction. This ensures that other transactions cannot see partial results. Since flat transaction usually end in a short amount of time locking resources for the duration of the transaction is feasible.

A typical B2B scenario such as a supply chain spans multiple companies that each plays a distinct role: manufacturer, supplier, logistics provider etc. In automated supply-chains, computer systems of multiple companies engage in inter-enterprise business transactions.

The supply chain transactions are long running and can last for minutes, hours, days, weeks or even years. In this scenario locking is clearly not a feasible approach to achieve isolation, since it would require companies to lock their databases while waiting for others to finish their part of the transaction. Therefore we cannot assume a single layer of control, as required by the flat transaction model: the transaction model should allow the main transaction to be broken into independent *sub-transactions*.
An extension of the flat transaction model that allows sub-transactions is the concept of *nested transactions*. In this model there are a number of *sub-transactions* (flat or nested) that make up one *main or top-level transaction*. The outcome of the sub-transactions is tied to the main transaction. If the main transaction aborts all sub-transactions must abort. For a detailed description of nested transactions refer to [3].
A special case of the nested transaction model is the *multi-level* transaction model. Multi-level transactions provide more flexibility in completion control than the basic nested transaction model. In this model it is assumed that participating resource managers can manage their own sub-transactions and can decide to pre-commit their sub-transactions before the main transaction completes. Pre-committing of the sub-transaction means that the resources release locks involved in the sub-transaction and the state is saved to durable storage. While multi-level transactions provide more flexibility in terms of sub-transaction management for resource managers it also introduces a problem with the completion of the main transaction. What happens if a sub-transaction has already pre-committed and the outcome of the main transaction is an abort? The solution is to apply a *compensating transaction* to the already committed sub-transaction. The compensating transaction can reverse all the actions made by the sub-transactions and implement the local aspect of the failure for the main transaction.
Based on the discussion above, we conclude that the multi-level transaction model can be successfully applied for B2B transactions. In the second half of this document we are going to use this model to define the Business Transaction Protocol.

BTP and two-phase commit

The standard method for achieving the ACID properties in short-running transactions involving multiple (possibly distributed) resources is the two-phase commit protocol. Since two-phase commit assumes the participating resources to be protected, it is not suitable for long-running B2B transactions, where resources are managed by systems that belong to separate companies. We suggest a different approach from two-phase commit, where participating resources are allowed to pre-commit their sub-transaction and apply a compensating action in case the main transaction terminates with a failure.

Scope of BTP

The goal of BTP is to manage the propagation of the result (success or failure) of the business transaction in a reliable way to all the involved resources. BTP *does not* specify the business protocol governing the business transaction. It merely provides facilities and semantics for a reliable termination mechanism to achieve a shared agreement on the outcome of the business transaction.
BTP alone cannot guarantee the atomicity, consistency and durability. The systems that participate the protocol have to manage their local resources accordingly to achieve these

attributes: e.g. on termination with failure they have to execute the appropriate compensating action.

BTP and other B2B protocols

BTP is “agnostic” regarding the underlying B2B protocol stack so it can be easily implemented in conjunction with other standards, such as ebXML or SOAP. For example, a header can be added to the ebXML message envelope to carry the transaction context defined by BTP. The system messages that are used by BTP, like *startTransaction* or *terminateTransaction* can be sent as standard ebXML messages.

Business Transaction Protocol

In this section we discuss the specifics of the proposed Business Transaction Protocol.

Concepts and terminology

This section introduces the key concepts and terminology that are used by BTP. We describe a model for the transaction protocol including the roles the different system components play during the lifecycle of the transaction.

Trading partner

A *trading partner* is a representation of an entity, such as a company, that participates in one or more business transactions. A trading partner has a server (*B2B server*), which hosts applications that exchange messages with other trading partners (*Note*: in this document we will use the terms *trading partner* and *trading partner application* interchangeably).

Transaction

A *transaction* is a series of message exchanges between a set of trading partners to implement a common business process.

Initiator and participants

A transaction is always initiated by an application of a trading partner (*initiator*). The applications of trading partners that take part in a transaction are called *participants*. The initiator is a special participant. The completion of a transaction can be either due to termination request issued by the initiating trading partner by the system if the transaction times out.

Transaction coordinator

The B2B server of a trading partner also runs a *transaction coordinator* component that implements the Business Transaction Protocol. It enlists and de-lists participants in a transaction and participates in the transaction termination protocol. Coordinators can play the role of a main or subordinate coordinator in a transaction. There is only one main coordinator in a transaction.

Main Coordinator

The main coordinator is the one that receives the *createTransaction* request from the initiator of the transaction. It also drives the termination protocol for that transaction.

Subordinate Coordinator

The subordinate coordinator, that cooperates with the main coordinator for terminating a transaction.

The life of a transaction

The life of a transaction consist of the following events:

- It is started by the initiator
- The initiator and the participants exchange messages
- Participants can leave the transaction
- The transaction can terminate with success, failure or timeout

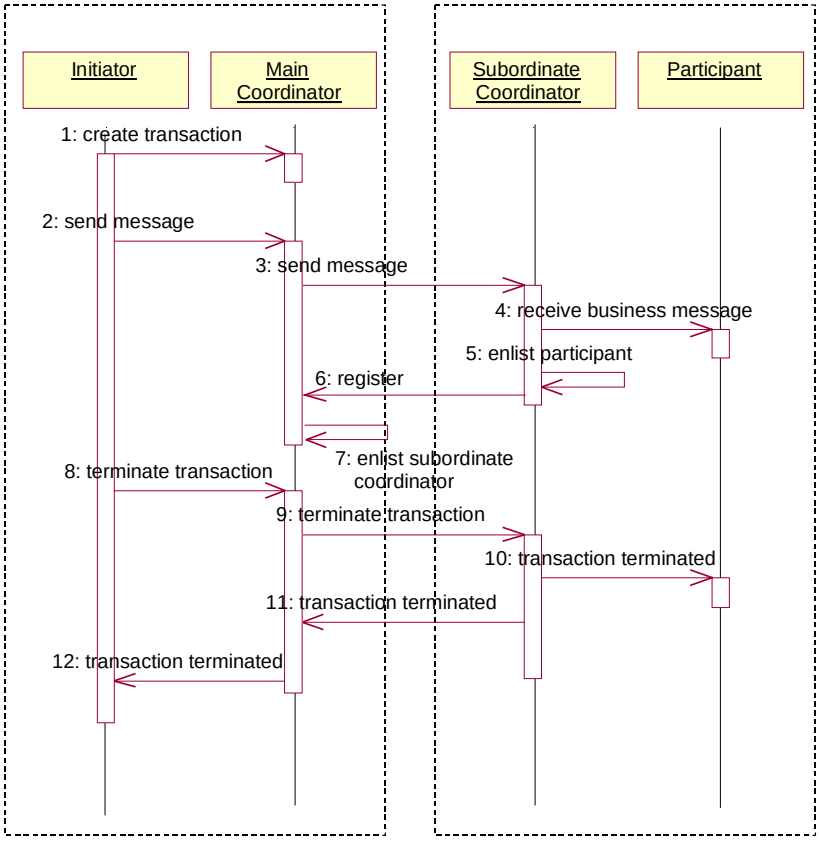

Figure 2 Transaction lifecycle

Starting a transaction

A transaction is created at the request of the initiator. When a transaction is created, it is assigned a globally unique id by the main coordinator. The transaction is considered to be in ACTIVE state. Once a transaction is in ACTIVE state, business messages can be exchanged in that transaction.
The selection of the main coordinator depends on the topology of the participants. For BTP we consider two kinds of topologies: point-to-point and hub-and-spoke.

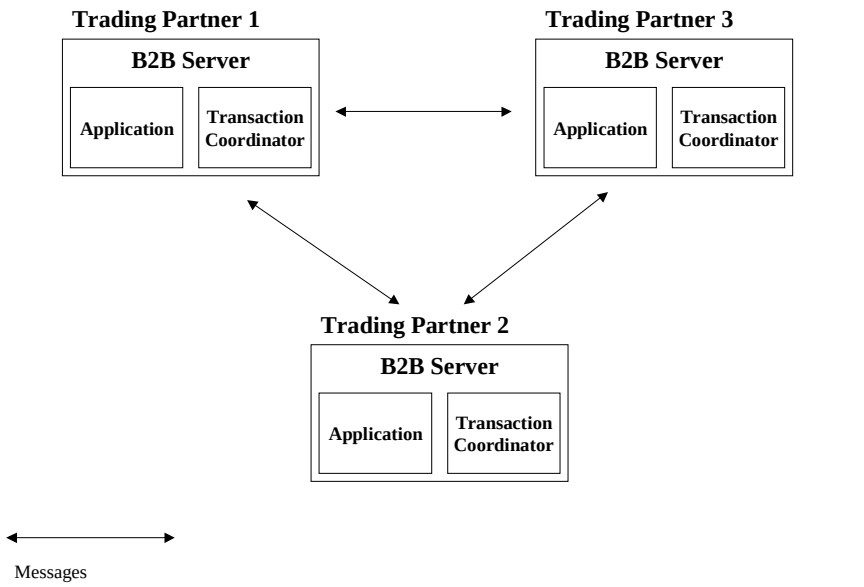

Figure 3

Figure 4 Point-to-point topology

In the point-to-point topology servers of trading partners exchange messages directly with each other. In this case, the coordinator in the initiator's system becomes the *main coordinator* for the transaction.

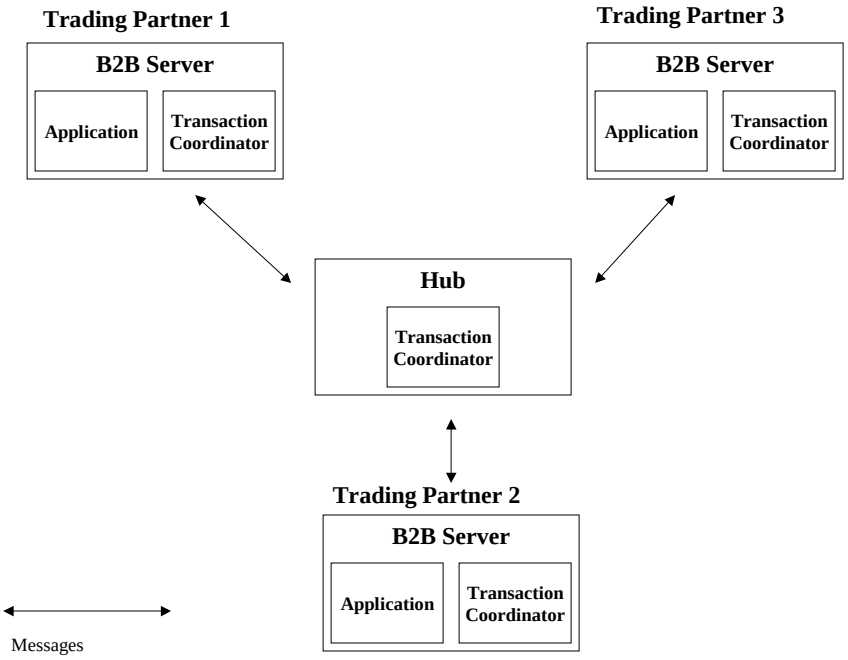

Figure 5 Hub-and-spoke topology

In the hub-and-spoke topology messages are exchanged via an intermediary, the hub. In this scenario the transaction coordinator in the hub is always the main coordinator.

Exchanging messages in a transaction

Messages are exchanged between participants in a transaction. Each message sent in a transaction carries a transaction context. The transaction context helps coordinators to identify the transaction instance to which each message belongs and to take appropriate action, e.g. enlist a participant in a transaction.

Transaction context

The transaction context consists of three components: transaction identifier, transaction type and the URL of the main coordinator of the transaction.

Transaction Identifier

The transaction identifier is a globally unique identifier. The main coordinator of the transaction generates this identifier.

Transaction Type

The transaction type defines the business transaction.

Transaction timeout

The timeout value defines the maximum amount of time the transaction should be active.

Main coordinator's URL

This contains the location information of the main coordinator of the transaction.

Transaction infection

Participants get “infected” by the transaction via receiving a message that carries the transaction context. Initially only the initiator is “infected”. As the initiator sends messages to other participants they become “infected”. In turn, “infected” participants can “infect” others by sending messages to them.

When a message reaches a trading partner’s server, the coordinator at the trading partner’s server intercepts that message and extracts the transaction context from it. If the transaction was unknown to the coordinator, it makes a note of the transaction by storing the transaction context. The message is then delivered to the recipient trading partner’s application for further processing. After delivering the message, the coordinator enlists the recipient trading partner’s application as a *participant* in the transaction. At this moment, this coordinator becomes a *subordinate coordinator* for this transaction. It then notifies the main coordinator of its involvement in the transaction by sending a *register* request. The register request contains the URL of the subordinate coordinator as well as the transaction context. The main coordinator then adds this coordinator as a subordinate coordinator for this transaction.
(see Future Work section for an alternate approach for subordinate coordinator registration)

Leaving a transaction

The participants other than the initiator can *leave* the transaction by notifying the subordinate coordinator that enlisted them. Leaving the transaction means that the participant will not be notified about the outcome of the transaction.

Transaction termination

The initiator is the only participant that is allowed to terminate the transaction. In order to terminate the transaction, the initiator sends a *terminate* request to the main coordinator. Then the main coordinator together with all the subordinate coordinators jointly executes the termination protocol. When the termination protocol starts, the transaction is put into TERMINATING state. After the termination protocol has completed, the transaction is put in the TERMINATED state. The coordinators do not allow messages to flow for a transaction while it is being terminated. In the next section we describe the termination protocol in detail.
A transaction can be terminated with success or with error. Transaction termination with error triggers the appropriate compensating transaction at the participating trading partners’ server. It is outside the scope of the Business Transaction Protocol to define how the compensating transaction is managed.

Transaction timeout

The initiator assigns a timeout value to each transaction. If a transaction times out while still ACTIVE, the main coordinator automatically executes the termination protocol with failure.

Termination Protocol

To describe the termination protocol, we have adopted the style used by Bernstein et. al. [ref 1, 2] for describing the two phase commit protocol. We first list the assumptions made while defining the protocol. Then we describe sequence of actions in the protocol. While discussing life cycle of the transaction, we briefly mentioned about the states of the transaction and the transaction context. We describe in details about the states and the context here.

Assumptions

The transaction termination protocol makes the following assumptions:

1. All the processes involved in termination get the same decision (terminate with success or failure) from the initiator directly or indirectly.
2. Only the initiator or the main coordinator (in case of a timeout) can terminate the transaction. A participant can only leave the transaction. The initiator can't leave the transaction.
3. Only one coordinator acts as a main coordinator for a transaction. A coordinator can act in main and subordinator roles simultaneously only for different transactions.
4. The coordinator fails by stopping. The protocol does not misbehave in case of malfunction of the system.
5. A coordinator maintains a persistent log and a transactional recovery system.
6. The underlying messaging system offers "exactly-once" delivery semantics as well as retries in case of delivery failure as required for the transaction termination messages.

Sequence of actions

The following actions take place while terminating a transaction:

1. The initiator issues a *terminate* request for the transaction. The main coordinator for this transaction receives the request, which contains the result of the transaction. The result can be success or failure.
2. The main coordinator first marks the transaction TERMINATING and then sends the *terminate* request to all subordinate coordinators for that transaction. Then it waits a pre-defined amount of time for the *termination completion* notification from the subordinate coordinators. If the main coordinator cannot notify a subordinate coordinator because of network or site failure, it marks the subordinate coordinator unreachable.
3. On receipt of the *terminate* request, the subordinate coordinator notifies all the enlisted participants of the transaction about the result of termination. If the result of the transaction was failure the participant can invoke the appropriate compensating transaction.
4. After it notified all its enlisted participants the subordinate coordinator sends *termination completion* notification to the main coordinator. This notification indicates that the subordinate coordinator has completed the process of terminating the transaction at its end.

5. Having received *termination completion* notification from all the subordinate coordinators, the main coordinator marks the transaction TERMINATED. If a subordinate coordinator does not send the termination completion to the main coordinator in the pre-defined time the main coordinator will re-send the termination request a pre-defined number of time before marking the subordinate coordinator unreachable.
6. The main coordinator then notifies the initiator of the completion of termination of the transaction

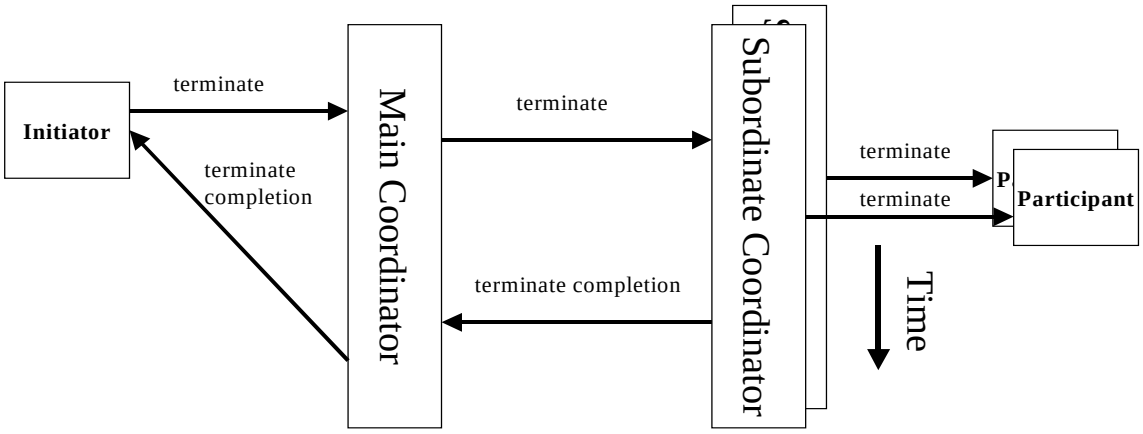

Figure 6 Transaction termination

Failure Handling and Recovery

Various kinds of failures can occur in a distributed system like a B2B commerce system. Two common failures are the communication link failure and the site failure. For reliable transaction coordination in such systems, a protocol such as the termination protocol of the Business Transaction Protocol is required.

Site failure

A site failure may be due to system failure. When a site fails, it stops processing and all the volatile data is lost. We assume that a site fails by stopping. This means that either it is operating correctly or not operating at all, it never operates incorrectly. A failed site is

recovered by executing a recovery procedure, which brings it to a consistent state ready for normal processing.

Communication Failures

The other cause of failures in a B2B system is a communication failure. The sites of B2B systems are connected through network links (the Internet). A communication failure is considered to have occurred when:

1. A message gets corrupted during communication between two sites.
2. A message is lost due to malfunctioning of a network link.
3. Two sites cannot communicate due to unavailability of a network path.

In the first two cases, we rely on network protocols to provide reliability. We assume that the network protocol implementation takes measures against message corruption by using appropriate error correction coding techniques. We also assume that the transport protocols and the networking infrastructure takes care of masking malfunctioning of network links by re-routing and re-transmitting packets.
The last case might occur due to network partition. A network partition can occur if a combination of sites and network links between the sites fail. Such failures can be avoided by designing networks with redundant communication paths. We take it for granted that such failures do occur and necessary actions are taken by the underlying messaging system infrastructure to mask against such failures.

Failure Detection by Timeout

We make an assumption that failure of a communication link or a site can be detected by a timeout in a distributed B2B system. As processes talk to each other by sending messages in such systems, a failure can be detected if the sender process does not receive expected reply within the timeout period. The failure to receive an expected message could be either due to a communication link failure or a site failure for the receiver. We also assume that the timeout period is calculated by taking into account events such as intermittent overloading of the communication network or the load on the process from which the message is expected.

Following are the cases when a coordinator or a participant is waiting for a message:

1. The main coordinator did not receive the *terminate* request from the initiator.
2. The main coordinator did not receive a *terminate completion* notification from a subordinate coordinator.
3. A participant did not receive the *terminate* request.

Timeout Actions

In the first case, the main coordinator can take advantage of the transaction timeout parameter, if it is applicable. If the initiator fails to terminate the transaction due to communication or site failure, and if the transaction life exceeds the timeout, the main coordinator terminates the transaction.

For the second case, if the main coordinator is not able to receive a *terminate completion* notification from one of the subordinate coordinators in time (implementation dependent parameter), it does not affect the overall result of the termination process. This is only an indication to the main coordinator to a keep record about the result of the transaction available, so if it later receives a request from that subordinate coordinator, it can answer with the result of the transaction.

In the third case, if a participant has not received the *terminate* request within the timeframe defined by the transaction timeout value, it can ask to leave the transaction. Here, if the transaction is still active, the coordinator will appropriately de-list the participant from the transaction. If the transaction is not active and the coordinator knows about the result of the transaction termination, it can reply with the result.

Recovery

To ensure the recovery of the main or subordinate coordinator in case of failure, it is essential to log sufficient information in reliable storage. The convention used to describe the logging is as follows:

“Eager” (or “forced” or “synchronous”) logging means log before proceeding to the next step in the protocol while “lazy” (or “asynchronous”) logging means the protocol can proceed before any actual logging takes place.

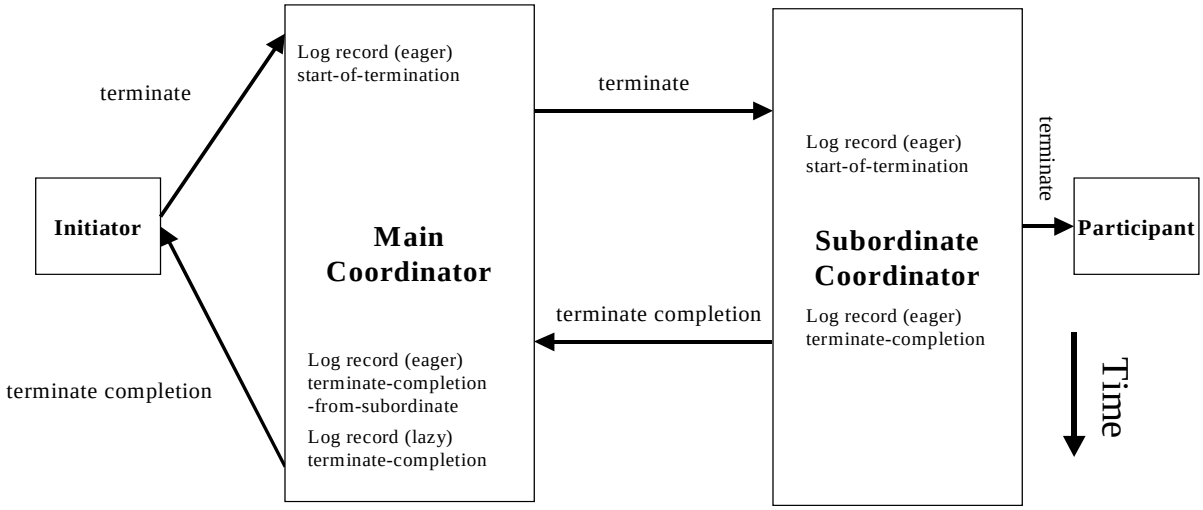

Figure 7 Logging

The main coordinator needs to write to the log for the following events as depicted on Figure 7 Logging:

1. When the main coordinator receives the *terminate* request from the initiator, it should (eagerly) log a start-of-termination record with the result of transaction (success or failure). This record also includes the transactions context and the list of subordinate coordinators infected with this transaction. It is an eager log because, if it fails after receiving the *terminate* request from the initiator, it would not know about the result of the transaction and the list of subordinate coordinators involved in the transaction.
2. On receiving *terminate completion* notification from each subordinate coordinator; it should (eagerly) log a record. This is an eager log because it records the outcome of the termination for the transaction at the subordinate coordinator. If this record is found on recovery for a subordinate coordinator, the main coordinator does not have to go through the process of re-sending the *terminate* request to that coordinator.
3. Once the last subordinate coordinator has sent the *terminate completion* notification (or has been marked unreachable), the main coordinator sends a formal *termination completion* notification to the initiator, marks the transaction **terminated** and logs the record. The record contains the timestamp when the transaction was marked terminated. This record can be logged lazily as it indicates that termination of transaction is completed at all subordinate coordinators. If this record was not logged

eagerly and on recovery if the log shows reception of *terminate completion* notification from all subordinate coordinators, the transaction is considered terminated.

The subordinate coordinator should log records as follows:

1. On receiving a *terminate* request from the main coordinator, it should eagerly write the record to the log. This record includes the result of the transaction (success or failure), the transaction context and all the participants that are enlisted for this transaction with this coordinator. This log should be eager because, otherwise the subordinate coordinator will have to contact the main coordinator (if available) to find out at least the result of the transaction on recovery.
2. After notifying the participants of the result of the transaction, it should eagerly log a record of termination completion before sending the *termination completion* notification for the transaction to the main coordinator. This record indicates completion of termination activities for the transaction. On recovery, if this record is not found in the log, the subordinate coordinator may unnecessarily have to contact the main coordinator of the transaction to know the outcome of the transaction and go through the process of probably re-notifying the participants.

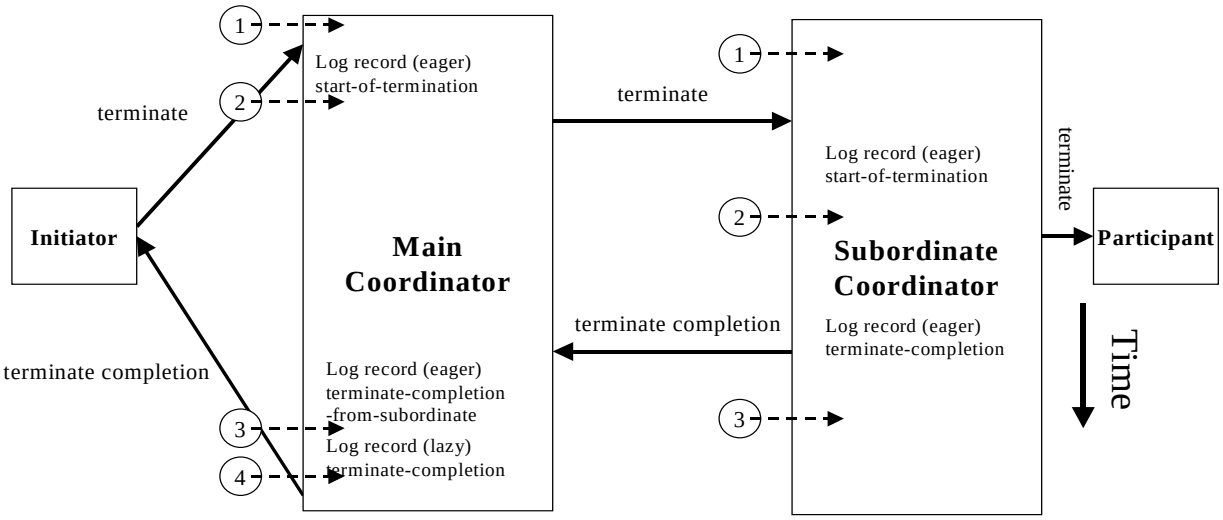

Figure 8 Transaction recovery

Let's examine how the logged information helps in recovering the main coordinator and the subordinate coordinator of a transaction.

The main coordinator of a transaction can be in one of the four states at recovery (Figure 8 Transaction recovery):

1. It has no start-of-termination record in the log on recovery. It did not receive *terminate* request from the initiator. The transaction is considered active by the main coordinator and all subordinate coordinators.
2. It has a start-of-termination record only. This means that the main coordinator either crashed after logging the start-of-termination record or before it received any *terminate completion* notifications from subordinate coordinators. It is possible that subordinate coordinators are waiting for the *terminate* request. The main coordinator sends a *terminate* request to all the subordinate coordinators. It is also possible that some or all of the subordinate coordinators have already received such a request. They will ignore the request.
3. It has terminate-completion-from-subordinate record(s) in the log. Here, there are two cases, either some of the subordinate coordinators have sent a *terminate completion* notification or all of the subordinate coordinators have sent it. In the first case, it is possible that some coordinators may still be waiting for the *terminate* request. The main coordinator will send a *terminate* request to those subordinate coordinators and wait for a *terminate completion* notification from them. If any of those subordinate coordinators, which were sent the request, have already received it earlier, they must ignore the subsequent requests. In the second case, the main coordinator will proceed to mark the transaction **terminated** and lazily log that record. It will then notify the initiator of the termination of the transaction. If the initiator has already received this notification, it must ignore it.
4. It has a terminate-completion record in the log. This means that the main coordinator has already marked the transaction **terminated**. Since in this case, the main coordinator has already completed its portion of the termination protocol it does not need to do anything further.

A subordinate coordinator of the transaction can be in any of the following 3 states (Figure 8 Transaction recovery):

1. It does not have a start-of-termination record in the log. In this case, it asks the main coordinator of the transaction to provide information about the outcome of the transaction by sending a *query status* request. If the main coordinator has no information about termination of the transaction, it must be waiting to receive *terminate* request from the initiator. The subordinate coordinator should also wait in that case for the *terminate* request from the main coordinator. If the main coordinator has information about the transaction termination decision, it will reply with the decision. The subordinate coordinator should then eagerly log the start-of-termination record and proceed.
2. It has a start-of-termination record. The subordinate coordinator should then notify participants of that transaction of the termination result. It should then proceed to log

(eagerly) a termination-completion record and then send the *terminate complete* notification to the main coordinator.

3. It has a terminate-completion record in the log. The subordinate coordinator has completed termination processing of the transaction locally. It is possible that the main coordination might still be waiting for *terminate completion* notification from this coordinator. It should send a *terminate completion* notification to the main coordinator if the timestamp in the record is not too long before recovery started (implementation dependent decision). The main coordinator can ignore it if it is a duplicate message.

Messages

This section describes the messages required for transaction coordination. We believe that these messages can be easily written in XML and sent using popular messaging protocols such as SOAP or ebXML TRP. We have assumed during the discussion of the protocol that a reliable messaging system is available for coordinating transactions. Messages required for transaction coordinator are described below.

Create Transaction

This message is sent from the initiator to the main coordinator. On receiving this message, the coordinator generates a globally unique identifier. A transaction is created in the system on behalf of the initiator. This coordinator assumes the role of the main coordinator for the newly created transaction.

Register Request

The subordinate coordinator must send a *register* request to the main coordinator of the transaction to register itself as a subordinate coordinator for that transaction. This request contains the transaction context and the URL where the subordinate coordinator can be reached.

Leave Transaction

The *leave* transaction message is sent from a participant to its locally accessible coordinator (subordinate coordinator) to indicate its intention to drop out from a transaction. This message contains the transaction context.

Unregister Request (optional)

When a subordinate coordinator receives a *leave* transaction message from a participant, it de-lists that trading partner's application from that transaction. It can optionally send *unregister* request to the main coordinator of that transaction at that time or it can choose to wait to receive a terminate request from the main coordinator. It sends *unregister* request, the request should contain the transaction context and the URL used for registration. If it decides to wait, it can immediately reply with a *terminate notification* when the *terminate* request eventually arrives.

Terminate Request

This message is sent to indicate termination of transaction. Different roles send it in different circumstances. For example, the initiator sends it to the main coordinator of a transaction to terminate that transaction. The main coordinator of the transaction sends it to subordinate coordinators of that transaction to indicate termination of transaction, and so on. This message contains the context of the transaction being terminated and result (success or error) of the transaction.

Terminate Completion Notification

The *terminate completion* notification is sent from a subordinate coordinator of a transaction to the main coordinator of that transaction notifying the main coordinator of completion of the termination protocol on its side. This message contains the transaction context and URL of the subordinate coordinator.

Query Status

This message is sent from the subordinate coordinator to the main coordinator on recovery to find out the status of the transaction.

Future Work

In this section we discuss some areas for future enhancements.

Vote from participants

The current proposal gives total control to the initiator participant regarding the outcome of the transaction. A possible enhancement is to provide all participants the ability to vote on the outcome of the transaction.

Associated transactions

Transaction could be nested within each to provide additional flexibility on the granularity of transactions.

Implicit registration of subordinate coordinators

In the current BTP proposal subordinate coordinators explicitly register themselves with the main coordinator, so all subordinate coordinators are known to the main coordinator. Alternatively, coordinators (the “parent”) could register other coordinators (“children”) when a message carrying the transaction context is first sent to them (form “parent” to “children”). This would result in a tree structure of coordinators with the main coordinator being the root. The termination algorithm would have to be modified so, that subordinate coordinators would have to act as the main coordinator for the coordinators registered with them.

References

1. Bernstein, P. A., V. Hadzilacos, and N. Goodman. 1987. “Concurrency Control and Recovery in Database Systems”. Reading, MA, Addison-Wesley.

2. Bernstein, P. A. and E. Newcomer. 1997. “Principles of Transaction Processing”. San Francisco, CA, Morgan Kaufmann.
3. Gray, J. and A. Reuter. 1993. “Transaction Processing: Concepts and Techniques”. San Francisco, CA, Morgan Kaufmann.