By Mark Little

TRANSACTIONS AND WEB SERVICES

Helping to realize the full potential of e-commerce.

WHILE WEB SERVICES HAVE EVOLVED AS A *means* to integrate processes and applications at an inter-enterprise level, traditional transaction semantics and protocols have proven to be inappropriate. Web services-based transactions (business transactions) differ from traditional transactions in that they execute over long periods, require commitments to the transaction to be "negotiated" at runtime, and isolation levels must be relaxed. Business transactions require an extended transaction model that builds on existing Web service standards and defines an interoperable transaction protocol and message flows that help negotiate transactions guarantees at the inter-enterprise level. The Organisation for Advance Structured Information Systems (OASIS) Business Transaction Protocol (BTP) is a protocol that meets the requirement for long-running collaborative business applications, allowing the relaxation of traditional ACID properties in a controlled manner specific to the Web services environment.

The concept of atomic transactions has played a fundamental role in creating today's enterprise application environments by providing guaranteed consistent outcome in complex multiparty business operations and a useful separation of concerns in applications. Rapid developments in Internet infrastructure and protocols have yielded a new type of application interoperation concept, making concepts that could only previously be considered in an abstract form an implementation reality [4]. The effects of such changes have been felt most strongly in business environments, fueling the mindset for extended transaction models better suited for Internet interoperation.

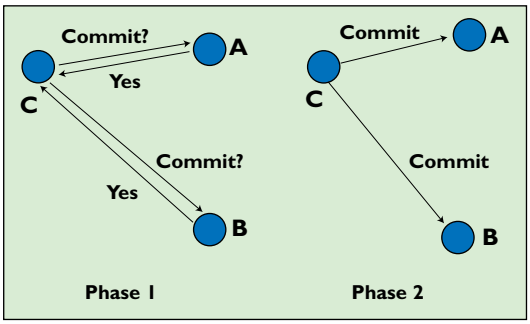

Figure 1. Two-phase commit protocol.

A business relationship is any distributed state maintained by two or more parties, which is subject to some contractual constraints agreed by those parties. A business transaction can therefore be considered as a consistent change in the state of a business relationship between parties. Each party in a business transaction holds its own application state corresponding to the business relationship with other parties in that transaction. During the course of a business transaction, this state may change. Since business relationships imply a level of value to the parties associated by those relationships, achieving some level of consensus among these parties is important.

In addition to understanding the outcomes, a participant within a business transaction may need to support provisional or tentative state changes during the course of the transaction. Such parties must also support the completion of a business transaction either through confirmation (final effect) or cancellation (counter effect). What it means to confirm or cancel work done within a business transaction will be for the participant to determine.

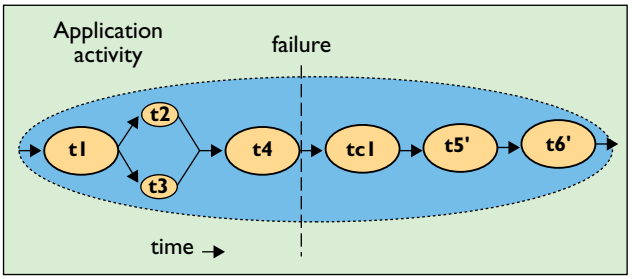

Figure 2. An example of a logical long-running "transaction," with failure.

Why ACID Transactions Are Too Strong. Atomic transactions are a well-known technique for guaranteeing consistency in the presence of failures [2]. Their ACID properties—Atomicity, Consistency, Isolation, Durability—ensure consistency of state is preserved, despite concurrent accesses and failures. Transactions are most suitably viewed as "short-lived" entities executing in a closely coupled environment, performing stable state changes to the system; they are less well suited for structuring "long-lived" application functions.

Traditional transaction systems use a two-phase protocol to achieve atomicity between participants, as illustrated in Figure 1. During the first (preparation) phase, an individual participant must make durable any state changes that occurred during the scope of the transaction, such that these changes can either be rolled back or committed later once the transaction outcome has been determined. Assuming no failures occurred during the first phase, in the second (commitment) phase participants may "overwrite" the original state with the state made durable during the first phase.

In order to guarantee consensus, two-phase commit is necessarily a blocking protocol: after returning the first phase response, each participant that returned a commit response must remain blocked until it has received the coordinator's phase 2 message. Until they receive this message, any resources used by the participant are unavailable for use by other transactions, since to do so may result in non-ACID behavior. If the coordinator fails before delivery of the second phase message these resources remain blocked until it recovers.

Therefore, structuring certain activities from long-running transactions can reduce the amount of concurrency within an application or (in the event of failures) require work to be performed again. For example, there are certain classes of application where it is known that resources acquired within a transaction can be released "early," rather than having to wait until the transaction terminates. In the event of the transaction rolling back, however, certain compensation activities may be necessary to restore the system to a consistent state.

Long-running activities can be structured as many independent, short-duration transactions, to form a "logical" long-running transaction [1]. This

structuring allows an activity to acquire and use resources for only the required duration of this long-running activity. This is illustrated in Figure 2, where an application activity (shown by the dotted ellipse) has been split into different, coordinated short-duration transactions. Assume the application activity is concerned with booking a taxi (t1), reserving a table at a restaurant (t2), reserving a seat at the theater (t3), and then booking a room at a hotel (t4), and so on. If all of these operations were performed as a single transaction then resources acquired during t1 would not be released until the transaction has terminated. If subsequent activities t2, t3… do not require those resources, then they will be needlessly unavailable to other clients.

However, if failures and concurrent access occur during the lifetime of these individual transactional activities then the behavior of the entire "logical long-running transaction" may not possess ACID properties. Therefore, some form of compensation may be required to attempt to return the state of the system to consistency [3]. For example, let us assume that t4 aborts. 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). Therefore, new activities are started; tc1, which is a compensation activity that will attempt to undo state changes performed, by say t2, and t3, which will continue the application once tc1 has completed. tc5' and tc6' are new activities that continue after compensation, for example, since it was not possible to reserve the theatre, restaurant, and hotel, it is decided to book tickets at the cinema.

LONG-RUNNING ACTIVITIES CAN BE STRUCTURED AS MANY INDEPENDENT, SHORT-DURATION TRANSACTIONS, TO FORM A "LOGICAL" LONG-RUNNING TRANSACTION.

The Business Transaction Protocol

The OASIS Business Transaction Protocol is for the coordination of autonomous parties involved in a business transaction.[1] With reference to Figure 3, the fundamental aspects of BTP are introduced here, followed by a description of how they can be applied within a Web services environment.

Consensus of Opinion. To ensure atomicity between multiple participants, BTP uses a two-phase completion protocol, with prepare, confirm, and cancel phases. Although this is similar to the two-phase commit protocol described earlier, BTP does not imply ACID transactions. How implementations of the prepare, confirm, and cancel phases are provided is a back-end implementation decision. Issues to do with consistency and isolation of data are also back-end choices and not imposed or assumed by BTP.

Open-Top Coordination. In a traditional transac-

[1]See www.oasis-opn.org/committees/business-transactions/.

SINCE A WELL-KNOWN SCHEMA IS USED TO DESCRIBE THE FORMAT OF BTP MESSAGES, THE MESSAGES PRODUCED BY DIFFERENT IMPLEMENTERS CAN BE ASSURED OF BEING EQUIVALENT.

tion system, the application or user has very few verbs with which to control transactions. Typically these are "begin," "commit," and "rollback." When an application asks for a transaction to commit, the coordinator will execute the entire two-phase protocol before returning an outcome (what BTP terms a closed-top commit protocol). The elapsed time between the execution of the first phase and the second phase is typically milliseconds to seconds.

However, the two-phase algorithm does not impose any restrictions on the time between executing the first and second phases. Therefore, BTP took the approach of allowing the time between the two phases to be set by the application by expanding the range of verbs available to include explicit control over both phases, that is, "prepare," "confirm," and "cancel"; what BTP terms an open-top commit protocol. The application has complete control over when transactions prepare and, using whatever business logic is required, later determine which transactions to confirm or cancel. This ability to explicitly control the termination protocol via business logic is a powerful tool since it supports a greater variety of strategies for implementing a transactional application.

Atoms and Cohesions. BTP introduced two types of extended transactions, both using the open-top completion protocol:

- Atom: the typical way in which transactional work performed on Web services is scoped. The outcome of an atom is guaranteed to be consistent, such that all enlisted participants will see the same outcome (cancel of confirm).
- Cohesion: this type of transaction was introduced in order to relax atomicity and allow for the selection of work to be confirmed or canceled based on higher-level business rules. A cohesion may give different outcomes to its participants such that some of them may confirm while the remainder cancel. The two-phase protocol for a cohesion is parameterized to allow a user to specify precisely which participants to prepare and which to cancel. The strategy underpinning cohesions is that they better model long-running business activities, where services enroll in atoms that represent specific units of work and as the business activity progresses, it may encounter conditions that allow it to cancel or prepare these units, with the caveat it may be many hours or days before the cohesion arrives at its confirm-set: the set of participants it requires to confirm in order for it to successfully terminate the business activity.

Once the confirm-set has been determined, the cohesion collapses down to being an atom: all members of the confirm-set see the same outcome.

XML Underpinnings of BTP. In BTP, an XML vocabulary is used for encoding messages conveyed between the participants in a transaction and the coordination infrastructure, as well as the format for describing transactional contexts. Since a well-known schema is used to describe the format of BTP messages, the messages produced by different implementers can be assured of being equivalent.

BTP Integration with the Web Services Stack. Though BTP does not define a complete network stack, its development was not undertaken in isolation from other work in distributed computing. As Web services are becoming widespread, the BTP committee was influenced in its decision making to a great extent by the emerging Web services architecture and protocols. To that end, the committee has defined a binding of the BTP message set to SOAP 1.1 over HTTP 1.1.

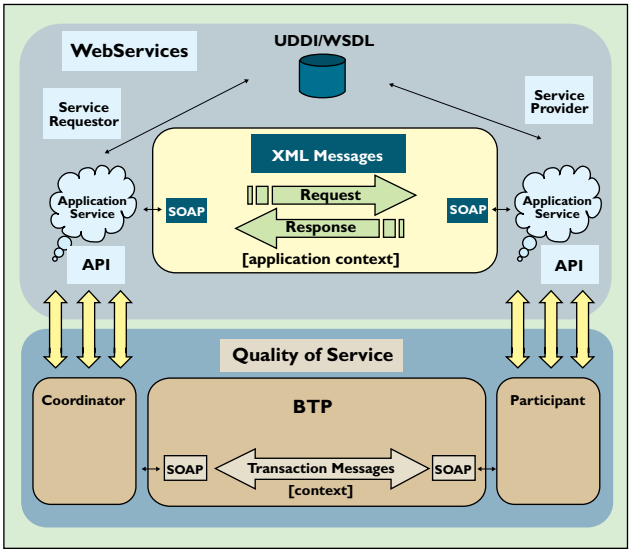

Figure 3. Web services, transactions, and contexts.

The BTP specification splits the messages into two types: those for the BTP infrastructure and those meant for the application. In situations where BTP messages are exchanged without the encumbrance of application messages, the strategy is straightforward: a message is simply propagated within the body of the SOAP envelope. For example, a typical begin message is represented as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP:Envelope
  SOAP:encodingStyle="http://schemas.xmlsoap.
  org/soap/encoding/
  "xmlns:SOAP="http://schemas.xmlsoap.
  org/soap/envelope/">
  <SOAP:Body>
       <btp:begin transaction-type="atom"
       xmlns:btp="urn:oasis:names:tc:
       BTP:1.0:core"/>
  </SOAP:Body>
</SOAP:Envelope>
```

Optimizations. Since BTP is intended for long-running transactions, it may be assumed performance has not been a prime factor in its development. However, this is not the case and in fact BTP contains a number of optimizations; we shall briefly describe two.

Resignation by Participants. In a two-phase commit protocol, in addition to indicating success or failure during the preparation phase, a participant can also return a "read-only" response; this indicates it does not control any work that has been modified during the course of the transaction and therefore does not need to be informed of the transaction outcome. This can allow the two-phase protocol to complete quickly since a second round of messages is not required. The equivalent of this in BTP is for a participant to resign from the transaction it was enrolled in. Resignation can occur at any time up to the point where the participant has prepared and is used by the participant to indicate it no longer has an interest in the outcome of the transaction.

Autonomous Decision by Participant. In a traditional two-phase protocol a participant enrolls with a transaction and waits for the termination protocol before it either confirms or cancels. To achieve consensus, it is necessarily a blocking protocol, which means that if a coordinator fails before delivering the final phase messages, prepared participants must remain blocked, holding onto (possibly valuable) resources. Modern transaction-processing systems have augmented two-phase commit with heuristics, which allow such participants to make unilateral decisions about whether they will commit or rollback. Obviously, if a participant makes a choice that turns out to be different from that taken by other participants, non-atomic behavior occurs.

BTP has its equivalent of heuristics, allowing participants to make unilateral decisions as well. However, unlike in other transaction implementations, the protocol allows a participant to give the coordinator prior knowledge of what that decision will be and when it will be taken. A participant may prepare and present the coordinator with some caveats as to how long it will remain in this state and into what state it will then migrate (for example, "will remain

prepared for 10 days and then will cancel the seat reservation"). This information may then be used by the coordinator to optimize message exchange.

So How Would You Use BTP?

Consider the situation where a user is booking a vacation, has provisionally reserved a flight ticket and taxi to the airport, and is now looking for travel insurance. The first group of work holds Flights and Taxi, since neither of these can occur independently. The user may then decide to visit multiple insurance sites (called A and B, in this example), and as he goes may reserve the quotes he likes. So, for example, A may quote $50, which is just within budget, but the user may want to try B just in case he can find a cheaper price and without losing the initial quote. If the quote from B is less than that from A, the user may cancel A, while confirming both the flights and the insurance from B.

How could we use BTP in order to coordinate this application in a reliable manner? The problem is that we wish to obtain the least-expensive insurance quote as we go along and without losing prior quotes until we know they are no longer the cheapest; at that point we will be able to release those quotes while maintaining the others. In a traditional transaction system, all of the work performed within a transaction must either be accepted or declined.

In BTP, however, we can use atoms and cohesions. A cohesion is first created to manage the overall business interactions. The application creates an atom (ReserveAtom, say) and enrolls it with the cohesion before invoking the airline and taxi reservation services within its scope. When a suitable flight and taxi can be obtained, prepare is called on ReserveAtom to reserve the bookings.

Then the insurance quotes are obtained by invoking their respective services within the scope of separate atoms (AtomQuote1 and AtomQuote2, for example), which are also enrolled within the controlling cohesion. When the quote from the first insurance site is obtained it is obviously not known whether it is the best quote, so the business logic can prepare AtomQuote1 to maintain the quote while it then communicates with the second insurance site. If that site does not offer a better quote, the application can cancel AtomQuote2 and it now has its final confirmation set of atoms (ReserveAtom and AtomQuote1), which it can confirm.

Relationship to Other Work

The Web services coordination (WS-C) specification aims to define a generic framework for coordinating abstract entities [5]. WS-C defines an abstract notion of coordination that is extended to specific coordination protocols by the addition of third-party agents. There is no equivalent of WS-C in BTP. While this means that WS-C is potentially more flexible that BTP because it can support a variety of coordination protocols, only one such protocol has been proposed: WS-Transaction [6]. WS-T has two transaction models:

- Atomic Transactions require ACID semantics and therefore assume that resources are locked for the transaction's duration; and
- Business activities are designed for use in long-running transactions. They ensure that any updates to state in a system are made immediately, significantly reducing the period during which locks must be held. WS-T has no notion of a two-phase commit for a business activity because commits are made immediately on receipt of the associated messages. If a failure occurs, a business activity runs compensating actions to restore data to a consistent form.

Conclusion

ACID transactions have proven invaluable over the years in the construction of enterprise applications. However, they are only really suited to short-duration activities executing on closely coupled applications and environments. When used in a loosely coupled environment like the Web, they prove too inflexible and restricting for many applications. The OASIS BTP has been developed to solve this problem while at the same time maintaining those aspects of the atomic transaction model that have proven useful. C

REFERENCES

1. Garcia-Molina, H. and Salem, K. Sagas. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data* (1987).
2. Gray, J.N. The transaction concept: virtues and limitations. In *Proceedings of the 7th VLDB Conference*, (Sept. 1981), 144–154.
3. Halliday, J.J., Shrivastava, S.K., and Wheater, S.M. Implementing support for work activity coordination within a distributed workflow system. In *Proceedings of the Third International Conference on Enterprise Distributed Object Computing (EDOC '99)*, (Sept. 1999), 116–123.
4. Little, M.C. et al. Constructing reliable Web applications using atomic actions. In *Proceedings of the Sixth Web Conference*, (Apr. 1997).
5. *Web Services Coordination (WS-C)*. Joint specification by IBM, Microsoft, and BEA, August 2002; www.ibm.com/developerworks/library/ws-coor/.
6. *Web Services Transactions (WS-T)*. Joint specification by IBM, Microsoft, and BEA, August 2002; www.ibm.com/developerworks/library/ws-transpec/.

MARK LITTLE (m.c.little@ncl.ac.uk) is a research fellow at The University of Newcastle upon Tyne, U.K.