Journal of Universal Computer Science, vol. 13, no. 8 (2007), 1094-1109
submitted: 15/1/07, accepted: 23/4/07, appeared: 28/8/07 © J.UCS

Achieving Atomicity for Web Services Using Commutativity of Actions

P. Michael Melliar-Smith
(Department of Electrical and Computer Engineering
University of California, Santa Barbara, USA
pmms@ece.ucsb.edu)

Louise E. Moser
(Department of Electrical and Computer Engineering
University of California, Santa Barbara, USA
moser@ece.ucsb.edu)

Abstract: Web Services enable the creation of complex business activities through the cooperation of independently developed software programs. However, Web Services incur the risk of long delays and locked data when using the classical distributed transaction strategy, and the risk of inconsistency when using the compensating transactions strategy. If the benefits of Web Services are to be fully realized, a better strategy must be employed. In this paper we describe an extended transactions strategy that can be used in conjunction with existing Web Services infrastructures, and that is compatible with existing business practices. The strategy exploits local transactions and commutativity of local actions at each data item to achieve global atomicity for business activities.

Keywords: Web Services, Distributed Transactions, Atomicity, Concurrency Control, Commutativity.

Categories: H.2.1 Information Systems, Database Management, Logical Design.

1 Introduction

During the past several decades, the great successes of computing and networking for enterprise applications have included:

- The Internet
- The World Wide Web
- Distributed Systems
- Transaction Processing
- Concurrency Control.

These technologies have come together as the foundation of Web Services [Champion et al. 2002] which, in the future, will dominate computer interactions within and between enterprises, and will facilitate cooperation and competition among enterprises.

Web Services enable the creation of complex business activities through the cooperation of independently designed and implemented software programs. To be able to couple together the computer systems of enterprises world-wide, so that they can cooperate without prior arrangements or prior knowledge of each other and without human intermediation, is a wonderful objective. Business activities based on Web Services will have major impacts on computing, business practices and international relationships. Such business activities must be reliable and must maintain data consistency, despite hardware and software faults.

The classical transaction processing paradigm [Gray and Reuter 1993], with its associated concurrency control [Bernstein et al. 1987], is effective in protecting business data against faults. It allows the construction of complex systems with efficiency, concurrency, reasonable development costs and acceptable levels of reliability. Each transaction can be designed separately, even though it is to be executed concurrently with other transactions. Moreover, a problem in one transaction can be recovered without adversely affecting other transactions. Most importantly, the transaction processing paradigm is easy to understand and to use.

However, classical transactions are quite centralized and are not well matched to the highly decentralized world of the Internet and the World Wide Web. Applying the traditional transaction processing paradigm in a wide-area distributed environment, with the XA Protocol [OpenGroup 1992] or the Web Services Atomic Transactions [Cabrera et al. 2005a], exposes the participants in the transactions to delays due to the locking of business data. Even without faults, the transactions might take quite a long time to complete, during which database records are locked.

If a transaction in one enterprise locks data in the database of another enterprise and then the server of that first enterprise fails, the data in the second enterprise might remain locked for an indeterminate period of time until the server in the first enterprise is recovered from the fault. The risk of such delays is unacceptable, particularly when the other participants in the business activity are unknown or of uncertain dependability. Consequently, in practice, Web Services Atomic Transactions are not used across a wide-area distributed environment.

The problems of distributed transactions, based on the two-phase commit protocol, can be reduced, but not eliminated, by use of the three-phase commit protocol [Skeen 1983]. However, the three-phase commit protocol increases transaction processing overhead and latency in the normal fault-free case. Consequently the three-phase commit protocol is not used in practice.

The Web Services Business Activity Specification [Cabrera et al. 2005b] addresses these problems by means of an extended transactions strategy with compensating transactions [Garcia-Molina and Salem 1987]. Compensating transactions are difficult to design and program, have a high error rate, and incur a high risk of leaving the databases in an inconsistent state. Detecting and removing such inconsistencies are difficult, labor intensive and time consuming, particularly when the inconsistencies span the databases of different enterprises.

We present below the results of our analysis of the probability that a business activity will leave the databases in an inconsistent state. For entirely reasonable values of parameters, that probability approaches unity. The designers and users of the Web Services Business Activity Specification appear to be unaware of the high risk of inconsistency resulting from compensating transactions.

The potential advantages of Web Services are impaired by the risk of long delays and locked data when using the distributed transaction strategy based on two-phase commit and conservative locking, and by the risk of inconsistency when using the extended transactions strategy with compensating transactions. If the benefits of Web Services are to be fully realized, a better strategy must be employed.

1.1 Business Activities and Transactions

We propose an alternative strategy for Web Services that is based on:

- Business activities that span multiple enterprises or sites
- Transactions that are strictly local to one site.

A *business activity* comprises a sequence of actions, typically but not necessarily at multiple sites. A business activity has a start and an end, and may involve multiple actions on the same data item.

An *action* is a part of a single business activity and is local to one site. A client program at one site invokes a Web Service to cause a server program at another site to perform a particular action. An action may involve one or more data items. The data items are also local to one site.

A *transaction* is a sequence of actions potentially but not necessarily at multiple sites. A transaction satisfies the classical ACID properties [Gray and Reuter 1993]. A *local transaction* is a transaction that involves a sequence of actions at one site that has no interactions with other sites. A local transaction also satisfies the classical ACID properties.

Figure 1 shows two business activities, each of which has actions that involve a single data item. Those actions are encapsulated in local transactions.

1.2 Preceding and Concluding Actions

For each data item, such as a database record, accessed by a business activity, the first action of the business activity on that data item is a *preceding action*. A preceding action imposes a constraint on the actions of the business activity, and also on the data item for the duration of the business activity.

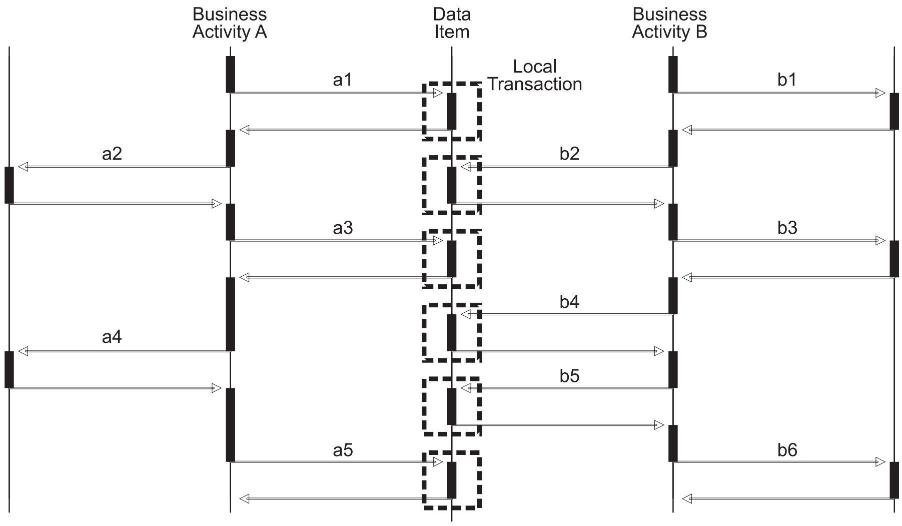


Figure 1: Two business activities, A and B. Business activity A invokes actions a1, a2, a3, a4 and a5 that are performed by multiple servers. Business activity B invokes actions b1, b2, b3, b4, b5 and b6 that are performed by multiple servers. Actions a1, a3, a5, b2, b4 and b5 all involve the same data item. The actions are encapsulated in local transactions.

The constraint imposed by the preceding action must ensure that the actions of the business activity on that data item commute with the actions of other business activities on that same data item. In [Weihl 1988], Weihl demonstrated that local commutativity of the actions of transactions on a data item suffices to maintain global atomicity for transactions, as illustrated in Figure 2.

The actions of a business activity on a data item are followed by a *concluding action* which, typically, records the effects of the actions and removes the constraints imposed by the preceding action.

The preceding and concluding actions are local to a single site and, indeed, to a single data item. A local sequential order on preceding actions on each data item individually suffices to guarantee commutativity and thus global atomicity.

The nature of the preceding action, and of the constraint imposed by that action, depend on the nature of the data item. For many business processes of the kind involved in Web Services, preceding actions and constraints are easily designed. For example, a constraint might limit the actions of the business activity to some specific amount of resources involving the data item. Correspondingly, the constraint might reduce the resources available to other business activities.

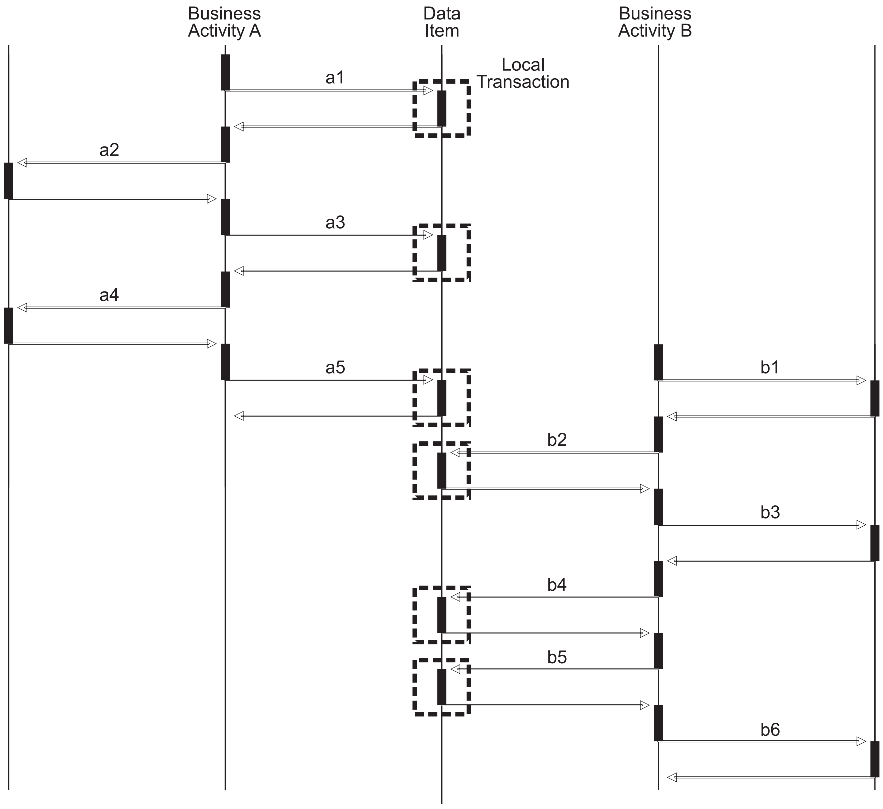


Figure 2: The commutativity of actions a1, a3 and a5 with actions b2, b4 and b5 is exploited to complete business activity A before starting business activity B, which illustrates how atomicity of business activities is provided by local commutativity of actions.

In effect, the resources have been allocated to, or reserved for, that particular business activity.

Commutativity, provided by the preceding and concluding actions, supports the separation of business activities and facilitates the design of reliable business activities and of recovery actions and abort actions. This commutativity-based strategy:

- Is easy to understand and to program
- Matches existing business practices
- Allows increased concurrency
- Provides higher reliability
- Extends existing database systems and Web Services infrastructures.

2 Implementing the Commutativity Strategy

There are two implementation methods for the preceding and concluding actions:

- An *implicit method* in which a client program invokes a server program to perform an action. The client program is unaware of the commutativity strategy used by the server program and of the preceding and concluding actions in the server program. The client program and the server program operate as if they were participating in a distributed transaction. The implicit method bears some resemblance to the escrow transactional method [O'Neil 1986].
- An *explicit method* in which a client program invokes a server program to perform an action. The client program is designed to accommodate the use of the commutativity strategy, typically a reservation strategy, by the server program and the reservation action that the server program employs. The explicit method exposes more of the mechanisms to the client program.

2.1 The Explicit Method

The description here of the explicit method is deliberately abstract because an implementation of the explicit method is rather tightly integrated into business logic mechanisms, such as reservations, and into the messages and protocols of the business logic, which are likely to be application specific. The explicit method allows application programs to include additional business logic mechanisms into the preceding and concluding actions, described below, such as a fee for a reservation. More details of a specific explicit method based on reservations can be found in [Zhao et al. 2005] and [Zhao et al. to appear].

We assume that a business activity has been started and has been assigned a unique identifier. The client program, as a part of the business activity, invokes an action to be performed by a server program in a request message that includes the identifier of the business activity.

The server program receives the request message to perform the action, extracts the business activity identifier from the request, and determines whether the server program is already participating in the business activity. If the server program is not yet participating in the business activity, the server program initiates a preceding action before executing the requested action. The preceding action also has a unique identifier, the reservation identifier.

The preceding action creates a record of the participation of the server program in the business activity. That record contains:

- The identifier of the business activity
- The identifier of the preceding action
- The reservations of resources that the preceding action has made on behalf of the business activity
- The actions the server program has performed for the business activity.

The preceding action then examines the request to perform the action and determines from the request which data items, or other resources, are involved in the request. The preceding action also determines which reservations of the data items or resources must be made to ensure that the action and other subsequent actions of the business activity will commute with the actions of other business activities. That commutativity is what ensures the atomicity and recoverability of the business activity, as Weihl demonstrated [Weihl 1988].

The preceding action records the reservations in the record it created for its participation in the business activity. It also records the reservations against the data items or resources it reserved, to ensure that other preceding actions cannot subsequently reserve those resources and that the actions of other business activities will commute with the actions of the business activity. Even though multiple business activities may proceed concurrently, and even though multiple actions may proceed concurrently, if several preceding actions involve the same data item or the same resource, those preceding actions must be executed sequentially, so that one preceding action can complete its reservations before another preceding action can make reservations that involve the same data items or resources.

The preceding action in the server program communicates with the client program that invoked the action, in order to offer the reservation. The server's offer may contain a time limit on the reservation. It may also contain a fee to be charged for the reservation. Alternatively, the server program informs the client program that insufficient resources are available and that the server program cannot perform the action requested by the client program. Another possibility is that the server program informs the client program that only some of the requested resources are available and can be reserved for it.

The client program communicates with the server program to accept the reservation. Alternatively, the client program might decline the reservation that the server program offered, in which case the server removes the reservation from the record and makes the resources available to other business activities or even to this business activity, if it makes a subsequent request for the resource. It is possible that, for some business activities, the acceptance is unnecessary, or that both the offer of the reservation and its acceptance are unnecessary.

At this point, the server program can invoke the action that the client program requested. Once the server program has performed the action, it records the action and also the resources that the action has consumed.

As part of the business activity, the client program might invoke the server program several times to perform actions that involve the same data item or resource, with each such request including the business activity identifier. Such actions are restricted to the resources that have been reserved for the business activity. If an action requires additional instances of a resource, the client program must make an additional reservation with a corresponding preceding action to reserve the additional resources before the action can be executed.

Once all of the actions associated with a particular preceding action have been invoked, the client program confirms the actions associated with the preceding action in the confirmation request, including the business activity identifier and the reservation identifier. The confirmation request may be combined with a request for an action on the data item or resource. Following the confirmation request, the business activity cannot invoke further actions on the data item or resource without another preceding action.

The client's confirmation request causes the server program to invoke a concluding action that determines which resources have been consumed by the actions of the business activity associated with the particular preceding action. Resources that have not been consumed are restored, so that they become available to subsequent actions of the same business activity or different business activities. Resources that have been consumed are removed, so that they are not available to subsequent actions of the same or different business activities. The record created to represent the participation of the server program in the business activity is deleted, or possibly logged for subsequent auditing or for fault recovery.

Alternatively, the client program might decide to cancel the actions associated with the preceding action. It then communicates that cancellation together with the activity identifier and the preceding action (reservation) identifier to the server program, which invokes the concluding action to cancel the actions associated with the preceding action. Another possibility is that the reservation time limits expire, in which case the server program also invokes a concluding action to cancel the actions associated with the preceding action.

The concluding action to cancel the actions associated with the preceding action determines, from the record for the business activity, which actions have been performed. The concluding action invokes actions to reverse any effects of actions subsequent to the preceding action. Because the actions commute, in general it is easy to design and program actions to reverse the effects of an action that has been taken. It is even possible that actions have no effects that must be reversed. Once the effects of the actions have been reversed, the concluding action deletes the reservations and restores the resources, so that they can be reserved by subsequent actions of the same or different business activities. The concluding action also deletes, or logs, the record that it had created for the business activity.

The commutativity strategy does not preclude deadlocks that might arise from the reservation of resources. Conventional deadlock detection, avoidance and recovery strategies can be exploited or, alternatively, resource allocation strategies specific to the business application might be employed.

2.2 The Implicit Method

In the implicit method, the client program is unaware of the use of the commutativity strategy by the server program and of the preceding and concluding actions in the server program. The client program and the server program operate as if they were participating in a distributed transaction, using the protocols and messages that they would use for such a distributed transaction. The implicit method does, however, depend on operations that can be made to commute. For example, operations that add or subtract values from a data field can commute, while operations that assign values to such a data field cannot commute with other similar operations on that data field.

When the client program sends a request message to the server program to perform an action, it includes in its request a business activity context. The business activity context contains a business activity identifier and the address of the business activity coordinator. For the implicit method, the behavior of the server program is substantially similar to the behavior described for the explicit method, except for the differences noted below.

When the server program receives the request to perform the action, it establishes reservations of resources substantially as described above. However, instead of communicating with the client program to offer a reservation, the server program communicates with the business activity coordinator to register its participation in the business activity. As part of that communication, it may inform the coordinator of a time limit on the duration of the business activity.

Within the business activity, the client program may invoke the server program several times to perform actions that involve a data item or resource, with each such request including the context of the business activity. If an action requires additional quantities of a resource, an additional reservation must be made before the action can be executed or, alternatively, the server program may communicate with the coordinator to cancel the reservation and abort the business activity.

When the coordinator completes the business activity, it communicates with each server that has registered its participation in the business activity. The server performs the concluding action and sends its confirmation to the coordinator. It is possible to use the classical two-phase commit protocol to complete the business activity, but that would expose the participants to a risk of delay and uncertainty in the event of coordinator failure. A one-phase commit protocol suffices if the granting of the reservation by the server and the registration of the server as a participant in the business activity is regarded as a promise by the server that the actions involving those resources and the concluding actions will be performed, as such a promise would be regarded in conventional business practice.

3 Evaluation of the Commutativity Strategy

We have performed several analyses to evaluate the commutativity strategy in comparison with the extended transactions strategy with compensating transactions, and the distributed transactions strategy based on two-phase commit and conservative locking. The analyses apply to both the explicit method and the implicit method, described above. More complete details of the performance analyses are given in [Zhao et al. to appear].

Figures 3(a) and 3(b) show the probabilities of potential database inconsistency for the commutativity strategy and the extended transactions strategy with compensating transactions. Concluding actions and compensating transactions are assumed to incur the same fault rate as regular transactions. The commutativity strategy has superior performance because there are fewer additional transactions for each fault recovery. It is our assessment that the difference in the probabilities that the databases are left in a potentially inconsistent state presents a decisive advantage for the commutativity strategy.

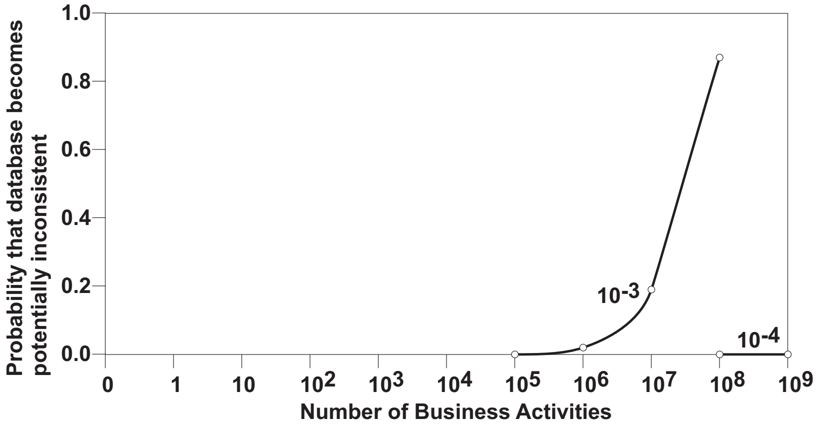
If the distributed transactions strategy based on two-phase commit and conservative locking is used instead of the extended transactions strategy with compensating transactions, the risk of inconsistency is reduced but there is an increased risk that data will be locked for an arbitrarily long period of time due to a fault.

We have also investigated the probability density functions (pdfs) for the duration of a business activity with delays due to lock contention, for both the commutativity strategy and the transactional locking strategy.

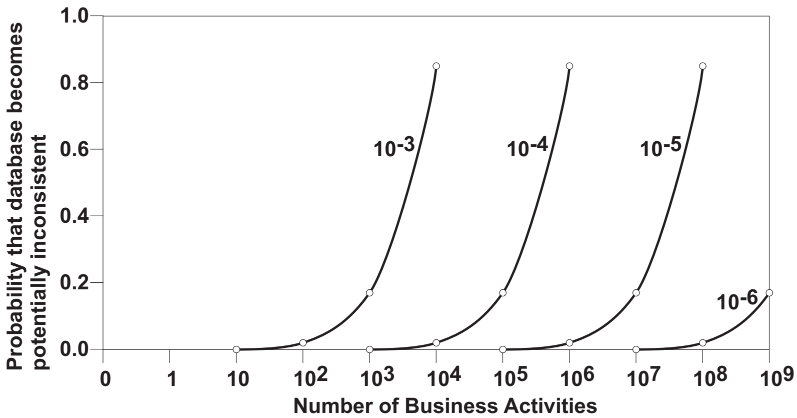
Figure 4 shows the pdfs for the duration of a business activity for different values of lock contention. When the probability of lock contention is low, the pdfs for the duration of a business activity using transactional locking are substantially as expected, and the effects of delays due to contention for a single lock and for two locks are clearly visible. As the probability of contention for a lock increases, the business activities are delayed, locks are held longer, delays due to lock contention are longer, and the probability that a business activity claims a lock that is already held by another business activity increases. The resulting pdfs have long tails and, thus, there is a high probability of lengthy delays for the business activity.

It is worth noting that, for transactional locking, there are probabilities for lock contention for which the system is not stable, representing unbounded delays and essentially no progress for the business activity. Such lock contention and instability leads to system collapse under heavy load.

Also shown in Figure 4 are pdfs for the duration of a business activity for the commutativity strategy. It is evident that even high probabilities of lock contention do not result in substantial delays for the business activity, because locks are held only briefly during the preceding and concluding actions and are



(a) Commutativity strategy.



(b) Compensating transactions strategy.

Figure 3: The probability that the databases are left in a potentially inconsistent state after increasing numbers of business activities for the commutativity strategy and the compensating transactions strategy.

not held for the full duration of the business activity. In summary, the commutativity strategy is more resilient to high loads and high probabilities of lock contention than the transactional locking strategy.

If, instead of distributed transactions based on conservative transactional locking, local transactions and compensating transactions are used, the concurrency achieved is substantially equivalent to the concurrency achieved for the commutativity strategy.

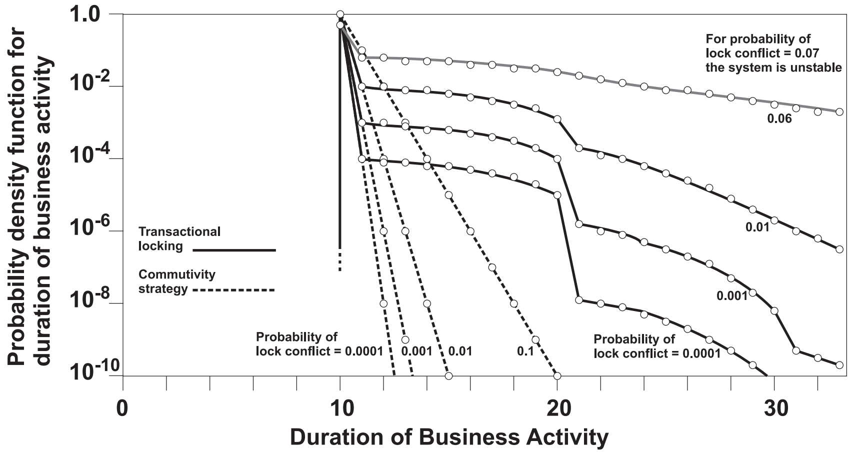


Figure 4: The probability density functions for the duration of a business activity with delays due to lock contention, for the commutativity strategy and the transactional locking strategy.

4 Related Work

Atomic transactions [Gray and Reuter 1993], with their associated concurrency control mechanisms [Bernstein et al. 1987], provide strong properties, the ACID properties, that allow the development of enterprise applications that are reliable and efficient. In [Weihl 1988] [Weihl 1989] [Fekete et al. 1990] [Weihl 1993], Weihl and his associates demonstrated that local commutativity of the actions of transactions on a data item suffices to maintain global atomicity for transactions. Our work is based on the work of Weihl.

Several researchers have investigated extended transaction models with the aim of providing greater flexibility and higher concurrency than traditional transactions with conservative concurrency control, by relaxing the traditional ACID properties.

The open nested transactions model [Weikum 1993] focuses on the compatibility of operations on abstract data types. If an operation on a particular data type is open, a new sphere of control can be spawned. Open nested transactions are naturally hierarchical and allow the results of a subtransaction to be viewed by other transactions before the top-level transaction is committed. Our strategy can be viewed as a specific kind of open nested transactions.

Multi-level transactions [Weikum 1991] are a special case of open nested transactions, where the transaction hierarchy is strictly layered. Likewise, the sagas strategy [Garcia-Molina and Salem 1987] can be characterized as a kind of open nested transactions. Sagas allow a long-running transaction to be executed as a number of ACID transactions. Sagas can see intermediate results of other sagas, and rely on compensating transactions to handle exceptions and to provide relaxed ACID properties for long-running transactions. The work of Garcia-Molina and Salem has lead directly to the strategies based on compensating transactions that we seek to improve.

In contrast, the traditional nested transactions [Lynch 1983] are closed nested transactions, where the sphere of control of a closed operation is its parent operation. Such transactions have the goal of ensuring atomicity, even if some of the operations fail, by exploiting the hierarchical structure.

The escrow transactional method [O'Neil 1986] bears some similarity to reservations and is one of the inspirations for our work. However, for distributed transactions, the escrow transactional method relies on the two-phase commit protocol. Moreover, it must be implemented as an integral part of a database system which, consequently, must understand the nature of the applications. Thus, its use has been limited.

The ConTract model [Wachter and Reuter 1992] is intended for defining and controlling business activities at a level above the level of atomic transactions. It addresses issues such as persistency, consistency, recovery, synchronization, cooperation, and the use of assertions as invariants on entry to and exit from business activities.

Other strategies [Barga et al. 2004] that provide stronger properties for long running transactions focus on recovery by ensuring that all operations are repeatable. Repeatability depends on strict concurrency control. Such strategies can be used in conjunction with the commutativity or reservation strategies.

Web Services standards have been developed to extend transactions into the domain of Web Services. The Atomic Transactions Specification [Cabrera et al. 2005a] provides classical distributed atomic transactions based on two-phase commit. The Business Activity Framework [Cabrera et al. 2005b] provides greater flexibility with the aim of supporting different kinds of business activities. The protocols in the Business Activity Framework are not explicitly two-phase, and depend on compensating transactions to handle exceptions and faults.

The Tentative Hold Protocol (THP) [Roberts and Srinivasan 2001] [Roberts et al. 2001], which has been considered for Web Services composition [Limthanmaphon and Zhang 2004], is used to exchange information between enterprises before a transaction begins. Unlike a reservation protocol, THP allows multiple clients to hold the same resource temporarily. It is possible that one client places an order for a resource when another client has already taken the resource. In this case, the business activity must be rolled back, and the client must apply a compensating transaction to cancel the committed transaction.

Other researchers [Fekete et al. 2003] [Greenfield et al. 2003a] [Greenfield et al. 2003b] have also focused on maintaining consistency in loosely-coupled distributed environments. They have proposed a language to express consistency conditions, tools to check whether the system maintains consistency, and guidance in using the infrastructure.

Related to a reservation protocol, but more expansive, is the protocol of [Ginis and Chandy 2000] for reserving resources in a free market. In their protocol, a consumer makes timed-reservation requests to the service providers in the form of purchasing options for using resources that the service providers supply. However, their protocol is not necessarily appropriate for implementing loosely-coupled, long-running business activities for which our strategy is intended.

5 Conclusion

The current direction of Web Services faces the danger of losing many of the advantages provided to enterprise computing by the ACID properties of atomic transactions. Business activities incur the risk of long delays and locked data when using the distributed transaction strategy based on two-phase commit and conservative locking, and the risk of inconsistency between databases when using the extended transactions strategy based on compensating transactions.

We have presented a strategy for achieving atomicity for loosely-coupled, long-running business activities. The strategy is derived from the work of Weihl, who demonstrated that local commutativity of operations on each data item suffices to maintain global atomicity for transactions. We have described an implicit protocol that can be hidden from the applications, and an explicit protocol that makes reservations of resources visible to the applications. The presented strategy can be used in conjunction with existing Web Services infrastructures, and is compatible with existing business practices.

References

[Barga et al. 2004] Barga, R., Lomet, D, Shegalov, G., Weikum, G.: "Recovery Guarantees for Internet Applications"; ACM Transactions on Internet Technology, 4 3 (2004) 289-328.

[Bernstein et al. 1987] Bernstein, P. A., Hadzilacos, V, Goodman, N.: "Concurrency Control and Recovery in Database Systems"; Addison Wesley (1987).

[Cabrera et al. 2005a] Cabrera, L. F., Copeland, G., Feingold, M., Freund, T., Johnson, J., Kaler, C., Klein, J., Langworthy, D., Nadalin, A., Orchard, D., Robinson, I., Storey, T., Thatte, S.: "Web Services Atomic Transactions"; http://download.boulder.ibm.com/ibmdl/pub/software/dw/library/WS-AtomicTransaction.pdf.

[Cabrera et al. 2005b] Cabrera, L. F., Copeland, G., Freund, T., Klein, J., Langworthy, D., Leymann, F., Robinson, I., Storey, T., Thatte, T.: "Web Services Business Activity Framework"; http://download.boulder.ibm.com/ibmdl /pub/software/dw/library/WS-BusinessActivity.pdf.

[Champion et al. 2002] Champion, M., Ferris, C., Newcomer, E., Orchard, D.: "Web Services Architecture"; http://www.w3c.org/TR/2002/WD-ws-arch-20021114/.

[Fekete et al. 2003] Fekete, A., Greenfield, P., Kuo, D., Jang, J.: "Transactions in Loosely Coupled Distributed Systems"; Proc 14th Australasian Database Conference, Adelaide, Australia (2003) 7-12.

[Fekete et al. 1990] Fekete, A., Lynch, N., Merritt, M., Weihl. W,: "Commutativity-Based Locking for Nested Transactions"; Journal of Computer and System Sciences, 41 (1990) 65-156.

[Garcia-Molina and Salem 1987] Garcia-Molina, H., Salem, K.: "Sagas"; Proc ACM SIGMOD Conference, San Francisco, CA (1987) 249-259.

[Ginis and Chandy 2000] Ginis, R., Chandy, K. M.: "Micro-Option: A Method for Optimal Selection and Atomic Reservation of Distributed Resources in a Free Market Environment"; Proc ACM Conference on Electronic Commerce, New York, NY (2000) 207-214.

[Gray and Reuter 1993] Gray, J., Reuter, A.: "Transaction Processing: Concepts and Techniques"; Morgan Kaufmann (1993).

[Greenfield et al. 2003a] Greenfield, P., Fekete, A., Jang, J., Kuo, D.: "Compensation Is Not Enough"; Proc 7th IEEE International Enterprise Distributed Object Computing Conference, Brisbane, Australia (2003) 232-239.

[Greenfield et al. 2003b] Greenfield, P., Fekete, A., Jang, J., Kuo, D.: "What Are the Consistency Requirements for B2B Systems?"; Proc High Performance Transaction Systems Workshop, Asilomar, CA (2003)

[Limthanmaphon and Zhang 2004] Limthanmaphon, B., Zhang, Y.: "Web Service Composition Transaction Management"; Proc 15th Australasian Database Conference, Conferences in Research and Practice in Information Technology, Dunedin, New Zealand (2004) 171-179.

[Lynch 1983] Lynch, N.: "Multilevel Atomicity - A New Correctness Criterion for Database Concurrency Control"; ACM Transactions on Database Systems, 8, 4 (1983) 484-502.

[O'Neil 1986] O'Neil, P. E.: "The Escrow Transactional Method"; ACM Transactions on Database Systems, 11, 4 (1986) 405-430.

[OpenGroup 1992] The Open Group: "Distributed TP: The XA Specification"; http://www.opengroup.org/public/pubs/catalog/c193.htm (1992).

[Roberts and Srinivasan 2001] Roberts, J., Srinivasan, K.: "Tentative Hold Protocol Part 1: White Paper"; http://www.w3.org/TR/tenthold-1.

[Roberts et al. 2001] Roberts, J., Collier, T., Malu, P., Srinivasan, K.: "Tentative Hold Protocol Part 2: Technical Specification"; http://www.w3.org/TR/tenthold-2.

[Skeen 1983] Skeen, D.: "A Formal Model of Crash Recovery in a Distributed System"; IEEE Transactions on Software Engineering, 9, 3 (1983) 219-228.

[Wachter and Reuter 1992] Wachter, H., Reuter, A.: "The ConTract Model"; Database Transaction Models for Advanced Applications, ed. Elmagarmid, A. K., Morgan Kaufmann (1992) 219-263.

[Weihl 1988] Weihl, W. E.: "Commutativity-Based Concurrency Control for Abstract Data Types"; IEEE Transactions on Computers, 37, 12 (1988) 1488-1505.

[Weihl 1989] Weihl, W. E.: "Local Atomicity Properties: Modular Concurrency Control for Abstract Data Types"; ACM Transactions on Programming Languages and Systems, 11, 2 (1989) 249-282.

[Weihl 1993] Weihl, W. E.: "The Impact of Recovery on Concurrency Control"; Journal of Computer and System Sciences, 47 (1993) 157-184.

[Weikum 1991] Weikum, G.: "Principles and Realization Strategies of Multilevel Transaction Management"; ACM Transactions on Database Systems, 16, 1 (1991) 132-180.

[Weikum 1993] Weikum, G.: "Extending Transaction Management to Capture More Consistency with Better Performance"; Proc 9th French Database Conference (1993).

[Zhao et al. 2005] Zhao, W., Moser, L. E., Melliar-Smith, P. M.: "A Reservation-Based Coordination Protocol for Web Services"; Proc IEEE International Conference on Web Services, Orlando, FL (2005) 49-56.

[Zhao et al. to appear] Zhao, W., Moser, L. E., Melliar-Smith, P. M.: "A Reservation-Based Extended Transaction Protocol"; IEEE Transactions on Parallel and Distributed Systems (to appear).