A Reservation-Based Coordination Protocol for Web Services *

Wenbing Zhao
Department of Electrical and Computer Engineering
Cleveland State University, Cleveland, OH 44115
wenbing@ieee.org

L. E. Moser and P. M. Melliar-Smith
Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106
moser@ece.ucsb.edu, pmms@ece.ucsb.edu

Index Terms: Business Activity, Continuous Availability, Extended Transaction Model, Relaxed Atomicity, Transaction Processing, Web Services

Abstract: Traditional transaction semantics are not appropriate for business activities that involve long-running transactions in a loosely-coupled distributed environment, in particular, for Web Services that operate between different enterprises over the Internet. In this paper we describe a novel reservation-based extended transaction protocol that can be used to coordinate such business activities. The protocol avoids the use of compensating transactions, which can result in undesirable effects. In our protocol, each task within a business activity is executed as two steps. The first step involves an explicit reservation of resources. The second step involves the confirmation or cancellation of the reservation. Each step is executed as a separate traditional short-running transaction. We show how our protocol can be implemented as a reservation protocol on top of the Web Services Transaction specification or, alternatively, as a coordination protocol on top of the Web Services Coordination specification.

1 Introduction

In enterprise applications based on Web Services [5], the computers of one enterprise can interact directly with the computers of other enterprises over the Internet. Such direct computer-to-computer interactions, without human supervision or intervention, provide speed improvements and cost reductions for distributed enterprise computing. However, such enterprise applications must operate with a high degree of availability and reliability. Problems in the operation of the application services can adversely affect the relationships between an enterprise and its customers, suppliers and partners. Moreover, the resolution of inconsistencies among the databases of multiple enterprises is difficult, expensive, time-consuming and error-prone, much more so than the resolution of inconsistencies within a database within a single enterprise.

Distributed transactions that span multiple enterprises need an atomic commitment protocol. The Two-Phase Commit (2PC) protocol [8] has been used successfully in many commercial transaction processing systems. However, the use of the 2PC protocol in distributed transactions unavoidably involves the locking of a data record of one enterprise by another enterprise. If the transaction coordinator fails, this locking period might be too long for an enterprise to tolerate. Even if the transaction coordinator does not fail, resources might be locked longer than an enterprise is willing to accept. Therefore, in practice, distributed transactions are seldom used for activities that span multiple enterprises. Instead, such business activities are based on extended transactions [6], where one or more localized transactions are run for each task and compensation transactions are applied to offset committed local transactions when a business activity is rolled back.

There are two problems with compensation transactions. First, compensating a committed task implies that one enterprise might retract what another enterprise committed, which is obviously undesirable. Second, before the compensating transaction is applied, another transaction might see the result of the committed task. Transactions that must be compensated are difficult to identify because there is no pre-defined way to find them. Furthermore, compensating transactions is very difficult, if not impossible. For example, if an end-of-quarter audit transaction is executed immediately after a sales task is committed in a publicly traded company, the sales are included in the total revenue reported to the public. The completion of the audit transaction, followed by compensation of the sales task, results in an inconsistency.

In this paper, we present a novel extended transaction protocol that avoids the use of compensating transactions while achieving atomicity and consistency similar to or better than existing extended transaction protocols. Each task

*This research was partially supported by MURI/AFOSR Contract F49620-00-1-0330 at the University of California, Santa Barbara, and a faculty startup award (for the first author) at Cleveland State University.

within a business activity is executed as two steps. The first step involves an explicit reservation of resources according to the business logic. In the interests of the supplier, a fee is associated with the reservation and the fee is proportional to the duration of the reservation. The second step involves the confirmation or cancellation of the reservation. Each step is executed as a separate traditional short-running transaction.

Our extended transaction protocol can be implemented either as a reservation protocol on top of the Web Services Transaction specification or as a coordination protocol on top of the Web Services Coordination specification.

The Web Services Transaction specification (WS-Tx) [3, 4] includes protocols for atomic distributed transaction commitment, as well as protocols for business activity coordination. The WS-Tx Business Agreement protocols are not two-phased, and exhibit some incompatibilities with our reservation protocol related to exception handling, as discussed later. However, like our reservation protocol, the WS-Tx Business Agreement protocols let the business logic determine the direction of the business activity (*i.e.,* rolling forward or rolling back).

The Web Services Coordination specification (WS-Coor) [2] describes an extensible framework for plugging in protocols that coordinate the actions of distributed applications. Such coordination protocols can be used to support a variety of applications, including those that require strict consistency and those that require agreement of a proper subset of the participants. The framework enables a Web Service to create a context needed to propagate an activity to other Web Services and to register for a particular coordination protocol.

2 System Model

A *Business Activity* (BA) is a unit of work that spans two or more enterprises and that consists of one or more tasks. A *task* is a short-duration unit of work that is executed as a traditional transaction within a single enterprise. The tasks in a BA are partially ordered. Tasks that are not causally related can be executed concurrently, and causally related tasks are executed according to the partial order.

Figure 1 shows an example business activity, highly simplified, of purchasing a product that requires shipping. The buyer proceeds with the purchase only if both the product and the shipping are available. Consequently, the buyer reserves the desired quantity of the product while negotiating the shipping. Each task shown in the figure is executed as a traditional transaction within a single enterprise.

A BA can be modeled as a *Business Transaction*, where there is a coordinator and multiple participants. For a Business Transaction, the traditional ACID semantics are not appropriate for the following reasons. First, it is not necessary that all of the participants see the same result. Second, the failure of a task does not need to result in the rollback of the entire Business Transaction. Third, the effect of executing a task might not be completely reversible, because of the business logic. In general, the commitment of a task implies that some form of service or work has been carried out. Consequently, there is a cost associated with the service or work, and of cancelling the task.

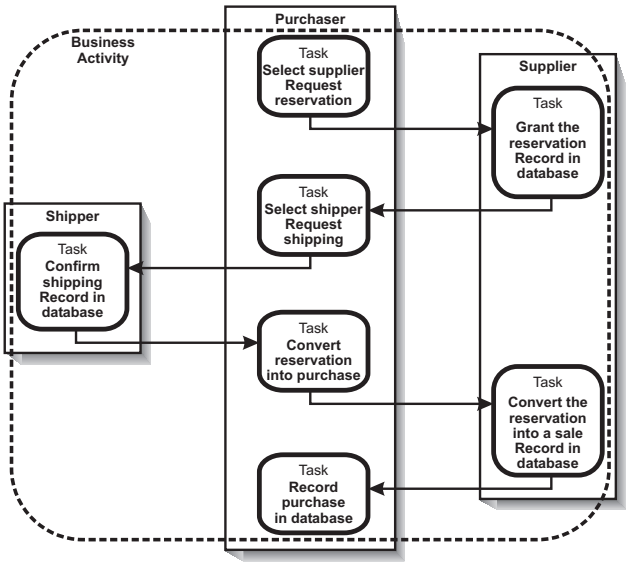

Figure 1. A business activity that spans multiple enterprises and comprises multiple tasks. Each of the tasks is executed as a traditional transaction within one of the enterprises.

A task can be modeled as an operation on one or more resources, which typically modifies some of the attributes of those resources. For example, the task of purchasing/selling certain kinds of goods (resources), such as automobiles, can be modeled as an operation on those resources in which the owner attribute of the resource is changed from the supplier to the buyer. An operation can be read-only, in which case, the task is a read-only task.

The coordinator and the participants in a business transaction are subject to crash faults but not arbitrary faults. The communication between the coordinator and participants in a business transaction is assumed to be reliable. We discuss the consequence of unreliable communication and how to work around that problem in Section 3.1.

3 Reservation-Based Coordination Protocol

Except for read-only tasks, each task is executed as two steps and is controlled directly by the application. In the first step, the resource involved in the task is reserved in a single transaction. In the second step, the reservation is either confirmed or cancelled according to the business rules,

also in a single traditional transaction.

To understand how a reservation is carried out, consider the example of purchasing goods (application-defined resources). The goods to be purchased are reserved at the seller upon a request by the buyer. For example, the attribute for the amount of goods to be sold shows "reserved", rather than "available" or "sold".

By explicitly reserving the resources involved in a task, the application has the flexibility of going forward or backward after the first step, without concern for the effects of the transaction, because a reservation action allows either outcome. Even if the intermediate result of the reservation is seen by other transactions, those transactions cannot make any assumptions about the future of the resources.

The coordination of different tasks within a business transaction is achieved by means of a two-phase protocol. In the first phase, the coordinator sends reservation requests to all of the participants, in an order determined by the business rules. Reservation requests for independent tasks can be sent concurrently. At the end of the first phase, the coordinator determines which reservations to confirm and which reservations to cancel. In the second phase, the coordinator sends the confirmation/cancellation requests to the appropriate participants.

In a traditional transaction, a fault at a participant might cause the rollback of the transaction, and the participant can decide unilaterally whether or not to abort the transaction. In contrast, in our reservation protocol, only the client is authorized to commit or rollback a business transaction. A fault at other participants might affect the client's decision and, thus, the outcome of the business transaction; however, it does not necessarily result in rollback.

3.1 Discussion

Resource reservation is a key concept of our coordination protocol. In the discussion below, we compare resource reservation with locking. Then, we discuss how our coordination protocol works under various fault scenarios.

Resource Reservation versus Locking. At an abstract level, resource reservation and (exclusive) locking of a resource are similar in that, in both cases, the resource is put on hold temporarily. However, resource reservation and locking differ significantly in the context of our coordination protocol and the concurrency control protocols used in database systems.

In our protocol, the reservation of a resource is executed as a traditional ACID transaction. The application has full control over the reservation activity and how long the resource should be reserved. In concurrency control for database systems, the locking of a resource is internal to the database system and is transparent to the application. The resource is locked by the database system, and the application has no control over how long the resource should be locked. In practice, typically a timeout controls how long a transaction lasts, and also prevents a resource from being locked too long. However, the timeout used in a database system is quite different from the duration of the reservation activity in our protocol.

Another difference between resource reservation and locking of resources is whether or not the owner of the resource can be paid for reserving the resource. In our coordination protocol, the reservation of a resource forms a contract between the client and the resource owner which explicitly reflects the fee that the client must pay. Moreover, the fee for the reservation can be determined in proportion to the duration of the reservation to discourage excessively long reservations of resources by clients. It is not obvious how to associate such a fee with the locking mechanism if it is internal to the database system.

Another difference between resource reservation and locking is the effect on other transactions that need to access the resource. If a resource is reserved and another transaction wants to access it, the transaction can acquire a lock on the resource, and the application can be immediately informed of the state of the resource (*i.e.,* it has been reserved). The application can thus take an appropriate action without delay. However, if the resource is locked and another transaction wants to access it, the transaction must wait until the lock is released. The waiting time might be long, in which case the application cannot take immediate action. To better understand this benefit of resource reservation, consider a seat on an airline flight as a resource. If the last seat has been reserved, another transaction that wants to reserve a seat can acquire the lock for that seat, look at the seat record, and immediately inform the application that there is no seat available. The application, or rather the client, might look for available seats on other airline flights. On the other hand, if the seat is locked, the late-coming transaction is simply blocked until the lock is released. When the transaction finally obtains the lock for the seat record, it provides the application with the same information, but at a (much) later time.

Operation under Fault Conditions. Under fault-free conditions, our reservation protocol ensures consistent results among the participants that have confirmed their tasks and, thus, it achieves a form of atomicity of the business transaction.

Consider what happens under fault conditions. The first phase of our protocol consists of reservation actions. Once a participant has granted a reservation, it is committed to the reserved resource until the client (through the coordinator) explicitly cancels the reservation. For this to work in the business world, a fee is associated with the reservation. To

encourage a buyer to decide quickly to confirm or cancel a reservation, the resource holder might use an exponentially increasing reservation fee, *i.e.,* an extended reservation costs much more than a short reservation.

In the second phase, if the coordinator wishes to confirm or cancel a reservation but it fails to communicate with a participant, all it must do is to take note of the current time and continue. Once a participant has granted a reservation, it is committed to honoring the reservation for the resource. The coordinator does not need to be concerned about the failure of a participant and can retain its current confirmation/cancellation. At the end of the second phase, the coordinator might re-attempt to communicate with the participants that were not reachable during the second phase.

If a failure occurs during the first phase, the effect is minor, *i.e.*, the failure limits the number of business partners from which the coordinator can choose. If a critical business partner is not available and an alternative cannot be found, the coordinator stops the first phase and starts the second phase by cancelling all existing reservations. In this case, the coordinator might still incur a cost for the reservations it made during the first phase, even though the entire business transaction has been rolled back. To avoid such a cost, the coordinator might make a reservation with a critical business partner first (knowing it does not have an alternative), or better yet, establish business relationships with other partners that provide similar services or goods in order to avoid a single point of failure.

Although we have assumed that a task within a business activity is a short-running traditional transaction, our protocol works equally well if a task is itself a business activity. That is, our protocol naturally supports the notion of scopes (parent-child relations) defined by the Web Services Transaction (WS-Tx) specification [3].

3.2 Example

Now we present an example of a business transaction that uses our reservation protocol. A client wishes to purchase a certain amount of a product from its suppliers and to arrange for the shipment of the product. There are two suppliers for the product and, because the two suppliers are geographically far apart, the client has to contact two different shipping companies that cover the shipping between the client and the two suppliers. First, we describe a scenario in which all of the resources are available and no fault occurs during the business transaction. Then, we discuss other scenarios.

At the beginning of the transaction, the client (*i.e.,* the initiator of the transaction) registers with the coordinator, and a coordinator object is created and returned to the client. During the transaction, the client conducts a number of read-only tasks such as a request-for-quote from each of the suppliers and shipping companies. Note that these steps are optional and the client might start by going directly to the reservation phase. After having retrieved enough information from its business partners, the client asks the coordinator to start the reservation phase by providing the resource name and endpoint information of each transaction participant. The coordinator then sends the reservation requests to the participants. Subsequently, all of the participants respond positively to the coordinator. The coordinator sends the reservation results to the client for selection. The client confirms the reservations for supplier 1 and shipping company 1, and cancels the reservations for supplier 2 and shipping company 2. Subsequently, the client informs the coordinator to start the second phase by providing a set of resources for which the reservations should be confirmed and a set of resources for which the reservations should be cancelled. The coordinator then sends confirmations to supplier 1 and shipping company 1, and sends cancellations to supplier 2 and shipping company 2. Again, all of the participants send replies to the coordinator to indicate that the requested tasks have been executed. The coordinator then informs the client that the business transaction has completed successfully.

In another scenario, supplier 1 rejects the reservation request and supplier 2 accepts the reservation request. Thus, the client is forced to use shipping company 2 because of geographical coverage. If shipping company 2 accepts the reservation request, the client confirms the reservations with supplier 2 and shipping company 2. If shipping company 1 accepts the reservation request, the client asks shipping company 1 (through the coordinator) to cancel the reservation. If shipping company 2 cannot satisfy the reservation request (for example, because there are not enough trucks available to ship the product on the required date), it rejects the reservation request and the client is forced either to find another shipping company or to rollback the business transaction. To rollback the business transaction, the client asks the coordinator to send a cancellation request to supplier 2.

If the coordinator cannot reach one or more participants during either of the two phases, the coordinator retries several times until a timeout occurs. If this happens during the reservation phase, it limits the choices the client can make, and might force the rollback of the transaction. Because a participant might crash immediately after it has accepted and handled a reservation request but before it sends the corresponding reply, the coordinator marks the participant and cancels the corresponding resource. If a participant becomes unavailable during the confirmation/cancellation phase, the coordinator reports an exception to the client. The client can then try to communicate with its partner using different communication channels, for example, telephones or faxes, and/or can establish a dedicated communication channel with that partner.

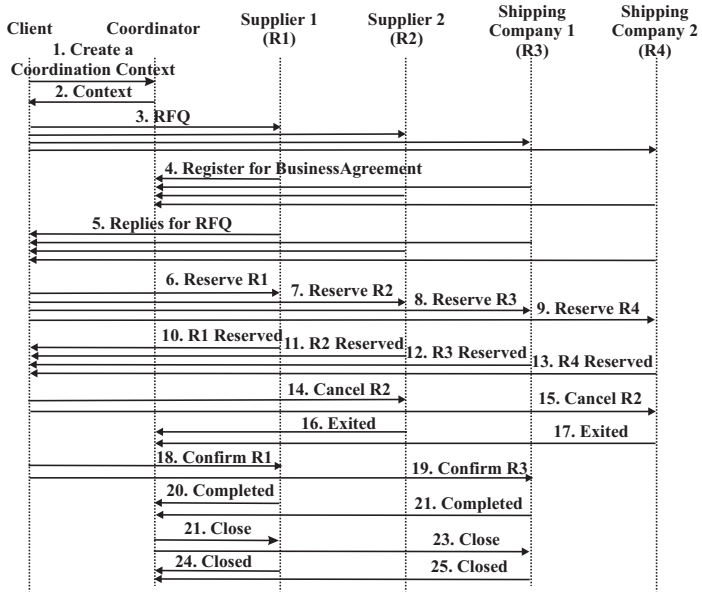

Figure 2. An example of a business transaction using the reservation protocol running on top of the WS-Tx Business Agreement protocol.

4 Implementation

Of particular interest is how the reservation protocol can be implemented on top of existing infrastructures and standards for Web Services. We have investigated two different directions: (1) how to coordinate a business activity using the reservation protocol on top of the business agreement protocol defined in the Web Services Transaction specification (WS-Tx), and (2) how to implement the reservation protocol as a coordination protocol for business activities similar to the protocols defined in WS-Tx, on top of the Web Services Coordination specification (WS-Coor).

4.1 Reservation Protocol on Top of WS-Tx

First we discuss how to use the business agreement protocol defined in the WS-Tx specification to coordinate a business activity according to our reservation protocol. We reuse the example business activity given in Section 3.2 for this purpose. We assume that the two suppliers and the two shipping companies provide the following Web Services: Request-for-Quote (RFQ), Reserve, Confirm and Cancel. We require that the coordinator is closely tied to the client process.

At the beginning of the transaction, the client (*i.e.,* the initiator of the transaction) registers with the coordinator and a coordination context is created and returned to the client, as illustrated in Figure 2, Steps 1 and 2. During the transaction, the client sends RFQ requests to the Web Services provided by the two suppliers and the two shipping companies (the transaction participants) (Step 3). Because the RFQ requests contain the coordination context for the business transaction, the transaction participants register with the coordinator (Step 4). The Web Services respond to the RFQ by sending the quotes to the client (Step 5). After retrieving enough information from its business partners, the client starts the reservation phase by sending the reservation requests to the four Web Services (Steps 6-9). Subsequently, all Web Services respond positively to the client (Steps 10-13). The client subsequently confirms the orders for supplier 1 and shipping company 1 and cancels the reservations for supplier 2 and shipping company 2. Subsequently, the client starts the second phase by sending cancellation requests to supplier 2 and shipping company 2 (Steps 14-15) and by sending confirmation requests to supplier 1 and shipping company 1 (Steps 18-19). As a result, the WS-Tx infrastructure at supplier 2 and shipping company 2 sends the Exit protocol message to the coordinator to resign from the business transaction (Steps 16-17). The WS-Tx infrastructure at supplier 1 and shipping company 1 sends the Complete protocol message to the coordinator to indicate that the tasks have been completed (Steps 20-21). The coordinator then sends Close protocol messages to the remaining two transaction participants (Steps 22-23), and the two participants respond by sending the Closed protocol messages to the coordinator (Steps 24-25).

The primary difficulty in implementing the reservation protocol using WS-Tx is that the coordinator would remove the participant that reported a fault (through a "Faulted" protocol message) from the participant list (the coordinator responds by sending a "Forget" message to the faulty participant) and the faulty participant would abandon the work it has done so far. This behavior is fine for our reservation protocol if the fault happens before the participant has committed the reservation. However, if a fault occurs after the participant has committed the reservation, the coordinator cannot abandon the reservation. Moreover, the fault should not cause the coordinator to remove the participant from the participant list, because only the client is authorized to confirm or cancel the reservation explicitly. Note that, as soon as a Web Service finishes handling the confirmation or cancellation request from the client, the corresponding participant sends a "Completed" message to the coordinator and transitions to the "Completed" state. A fault while a participant is in the "Completed" state does not generate a "Faulted" message.

The above behavior of the WS-Tx protocol can be modified by inserting a software layer into the WS-Tx infrastructure such that:

- At a participant, the detection of a fault and the sending of a "Faulted" message does not lead to abandoning committed tasks. Obviously, if there is a fault in the middle of a traditional transaction, the transaction is

aborted.

- On receiving a "Faulted" message, before taking any further action, the coordinator notifies the client application regarding the exception on the faulty participant, and the coordinator must wait until the client application has fully handled the exception. When a client receives an exception notification, the way in which it responds depends on whether it is in the first or second phase of the reservation protocol. If the exception occurs during the first phase, the client cancels the reservation with the faulty participant if the participant has already committed the reservation. The client can decide to rollback the entire business activity if the faulty participant is holding a critical resource, in which case the client stops the first phase and starts the second phase by cancelling all committed reservations. If the exception occurs during the second phase, the client cancels the earlier reservation placed with the faulty participant and confirms the reservation with another participant if such a participant is available. If this is impossible, the client retries the confirmation request with the faulty participant until it is eventually committed successfully.

In practice, it is desirable for a participant to mask or handle properly a transient fault by using existing fault tolerance technology such as replication, checkpointing, logging and replay [13]. The fee-based reservation policy also encourages an enterprise to ensure that its Web Services are highly available.

4.2 Reservation Protocol on Top of WS-Coor

Now we describe the reservation protocol as a Web Services coordination protocol on top of the WS-Coor framework. We omit the technical details of the XML schemas, the related WSDL declarations, and the content of the business activity CoordinationContext.

The coordinator-participant two-party state diagram of the reservation protocol is illustrated in Figure 3. The state of an individual participant in a business activity, on receiving and sending protocol messages is shown in the figure.

The coordinator sends the Cancel, Close, Forget, Exited, ReserveOrder, ConfirmOrder and CancelOrder messages. The participant sends the Register, Cancelled, Closed, Faulted, Exit, ReservedOrder, ConfirmedOrder and CancelledOrder messages. The ReserveOrder, ReservedOrder, ConfirmOrder, ConfirmedOrder, CancelOrder and CancelledOrder messages are not pure protocol messages, *i.e.,* they carry application payloads that must be delivered to the appropriate applications. The message types and the states are described below.

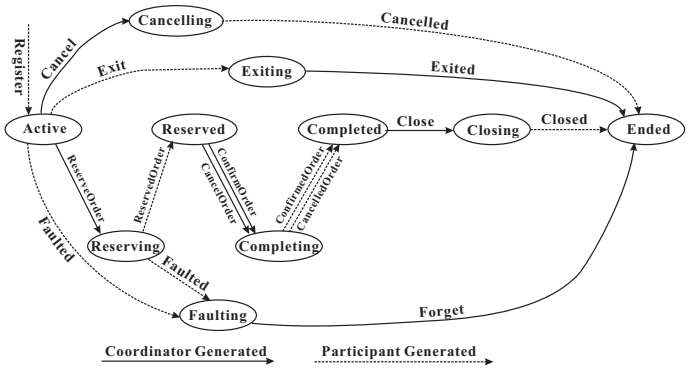

Figure 3. State diagram of the reservation protocol for Web Services.

- Cancel/Cancelled - The coordinator can remove a participant from a business activity when it is in the Active state by sending the participant a Cancel message. On receiving such a message, if the participant is in the correct state, it transitions to the cancelling state. The participant transitions to the Ended state when it sends a Cancelled message to the coordinator.

- Exit/Exited - A participant can leave a business activity by sending the coordinator an Exit message. Subsequently, it transitions to the Exiting state. On receiving an Exit message, the coordinator removes the corresponding participant from the business activity and responds with an Exited message. The participant transitions to the Ended state when it receives the Exited message sent by the coordinator.

- Faulted/Forget - A participant can encounter a fault while it is in the Active or Reserving state. It sends a Faulted message to the coordinator reporting the fault. On receiving a Faulted message, the coordinator invokes an exception handler registered by the application so that the application becomes aware of such a fault. When the exception handling is completed, the coordinator removes the participant from the business activity and sends the participant a Forget message. The faulty participant transitions to the Ended state when it receives the Forget message.

- ReserveOrder/ReservedOrder - The coordinator sends a ReserveOrder message (with application payload to provide details regarding the order) when the client indicates that it wishes to place a reservation order for a participant. On receiving the ReserveOrder message, a participant transitions to the Reserving state and passes the message to the application so that the order can be fulfilled. When the application indicates that the reservation order has been completed, the participant sends

a ReservedOrder message to the coordinator. (Note that the reservation request might be accepted or declined. Such information is included in the user payload portion of the ReservedOrder message.) On sending the ReservedOrder message, the participant transitions to the Reserved state.

- ConfirmOrder/ConfirmedOrder - The coordinator sends a ConfirmOrder message to a participant when the client indicates that it wishes to convert the reservation into a permanent order. On receiving the ConfirmOrder message, a participant transitions to the Completing state and passes the message to the application. When the application finishes converting the order, typically as a separate short-duration traditional transaction, the participant sends a ConfirmedOrder message to the coordinator and transitions to the Completed state.

- CancelOrder/CancelledOrder - The coordinator sends a CancelOrder message to a participant when the client indicates it wishes to cancel the reservation placed earlier. On receiving the CancelOrder message, a participant transitions to the Completing state and passes the message to the application. When the application finishes removing the previous reservation, the participant sends a CancelledOrder message to the coordinator and transitions to the Completed state.

- Close/Closed - When the two-phase completion protocol has been carried out, the coordinator sends a Close message to each of the participants. On receiving the Close message, a participant transitions to the Closing state and responds with a Closed message. The participant subsequently transitions to the Ended state.

Like the WS-Tx protocols, our reservation-based protocol requires that the coordinator and the participants are capable of handling duplicate messages.

5 Related Work

Broadly speaking, our reservation protocol is related to the extended transaction model and associated concurrency control mechanisms [1, 6, 8]. The concept of reservation has been widely used in practical systems that support business services, such as hotel, airline and automobile reservations. We have extended the concept of reservation in the coordination protocol for business transactions.

The most closely related work to our reservation protocol is the Business Transaction Protocol (BTP) [10] of the Organization for the Advancement of Structured Information Standards (OASIS), in particular, the BTP cohesion protocol. The BTP cohesion protocol is a two-phase protocol, in which the business transaction participants have explicit control over the two phases. In the first phase, all of the transaction participants are required to prepare, *i.e.,* they must ensure that the task can be committed or rolled back if a fault occurs. In the second phase, the transaction coordinator issues confirmation or cancellation requests to the participants. It is possible that only some of the participants have consistent results. One might argue that our reservation step is a special form of the prepare action and, thus, that our protocol is a special implementation of the BTP cohesion protocol. However, BTP only mentions a reservation-like action as one of several ways to provide provisional or tentative state changes. Moreover, the BTP specification does not pursue the concept of reservation to the same extent as we do, and it does not elaborate the benefits of the reservation approach.

Another closely related work is the atomic reservation protocol for reserving resources in a free market described in [7]. In the atomic reservation protocol, a consumer makes timed-reservation requests to the service providers in the form of purchasing options for using resources provided by the service providers. In the second phase, the confirmation/cancellation requests are sent to the service providers. Compensation transactions are used to wipe out the effect of a partially fulfilled plan and start with the next plan and to cope with failures and the expiration of options acquired during the first phase. The duration of the start of the first phase and the completion of the second phase are assumed to be short (on the order of a few seconds or minutes). Thus, the term "micro-option" is used to refer to the options acquired during the first phase. The atomic reservation protocol is not necessarily appropriate for use in loosely-coupled long-running business activities, for which our reservation protocol is specifically designed.

The tentative holding protocol (THP) [11, 12], which has been considered for Web Services composition [9], is used to exchange information between enterprises before a transaction begins. THP allows tentative non-blocking holds (reservations to be requested) for a business resource. The primary benefit of THP is to minimize the possibility of "rolling back" committed transactions by providing the client more accurate information regarding the availability of the business resource. Unlike our reservation protocol, THP allows multiple clients to hold the same resource temporarily. When one of the clients places an order, the remaining clients receive notifications of the unavailability of the resource. However, nothing prevents a client from placing an order for a resource immediately, at which point, another client might have already taken the resource. In such a case, business rules require the rollback of the business activity, and the client must apply a compensation transaction to cancel the previously committed transaction. In our reservation protocol, the reservation phase is part of the business transaction and the reservation is granted exclusively to a

client. Blocking reservations are used to avoid the need for compensating transactions.

Last but not least, it is interesting to compare our reservation protocol with the two-phase commit protocol [8] for atomic commitment of distributed transactions. Both protocols use two phases in which the first phase involves an exclusive blocking hold of a resource, and the second phase involves confirmation or cancellation of the reservation placed in the first phase. However, there are more differences than similarities between the two protocols. Our reservation protocol requires the holding of resources in terms of a traditional atomic transaction, which does not prevent other transactions from being executed. To compensate the blocking hold of the resources, we use fee-based reservations, *i.e.,* the client must pay a fee to the resource provider, based on the length of the reservation. Furthermore, our reservation protocol allows a proper subset of the participants to see a consistent outcome, while the two-phase commit protocol requires all participants to see the same outcome.

6 Conclusion and Future Work

We have described a reservation-based extended transaction protocol that can be used to coordinate the execution of business activities across multiple enterprises and that avoids the use of compensating transactions. The protocol comprises two steps. The first step involves an exclusive blocking reservation of the resource. The second step involves the confirmation or cancellation of the reservation. We have shown how to coordinate a business activity using the reservation protocol on top of the business agreement protocol defined by the Web Services Transaction specification and also how to implement the reservation protocol as a coordination protocol for business activities on top of the Web Services Coordination specification.

We plan to carry out future work along three directions. First, we plan to refine the fee-based reservation protocol so that it is more in line with the practice of reservations in the business world. For example, one can reserve an airline seat or a hotel room for some amount of time without a fee. Another issue is how to handle a reservation for a business that imposes a deadline if the client does not confirm or cancel the reservation before the deadline. Second, we plan to investigate the mechanisms necessary to facilitate application-level resource reservation and discovery. Our reservation-based protocol favors a resource-centric approach. Integrating our resource-centric protocol into a service-oriented architecture is an interesting and challenging problem. Third, we plan to implement a prototype system that incorporates our reservation-based extended transaction protocol with the current Web Services standards. Many new engineering problems will exhibit themselves when we build the prototype system.

References

[1] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison Wesley, February 1987.

[2] L. F. Cabrera, G. Copeland, W. Cox, M. Feingold, T. Freund, J. Johnson, C. Kaler, J. Klein, D. Langworthy, A. Nadalin, D. Orchard, I. Robinson, J. Shewchuk, and T. Storey. Web Services Coordination. September 2003. http://www.ibm.com/developerworks/library/ws-coor/.

[3] L. F. Cabrera, G. Copeland, B. Cox, T. Freund, J. Klein, T. Storey, and S. Thatte. Web Services Transaction. August 2002. http://www.ibm.com/developerworks/library/ws-transpec/.

[4] L. F. Cabrera, G. Copeland, B. Cox, T. Freund, J. Klein, D. Langworthy, I. Robinson, T. Storey, and S. Thatte. Web Services Business Activity Framework. Januaury 2004. http://www.ibm.com/developerworks/library/ws-busact/.

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

[6] H. Garcia-Molina and K. Salem. Sagas. In *Proceedings of the SIGMOD Conference*, San Francisco, CA, May 1987, pp. 249–259.

[7] R. Ginis and K. M. Chandy. Micro-option: A method for optimal selection and atomic reservation of distributed resources in a free market environment. In *Proceedings of the ACM Conference on Electronic Commerce*, New York, NY, October 2000, pp. 207–214.

[8] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[9] B. Limthanmaphon and Y. Zhang. Web service composition transaction management. In *Proceedings of the 15th Australasian Database Conference, Conferences in Research and Practice in Information Technology*, vol. 27, Dunedin, New Zealand, 2004, pp. 171–179.

[10] Organization for the Advancement of Structured Information Standards (OASIS). Business Transaction Protocol, version 1.0. May 2002. http://www.oasis-open.org/committees/businesstransactions/.

[11] J. Roberts, T. Collier, P. Malu, and K. Srinivasan. Tentative hold protocol part 2: Technical specification. November 2001. http://www.w3.org/TR/tenthold-2.

[12] J. Roberts and K. Srinivasan. Tentative hold protocol part 1: White paper. November 2001. http://www.w3.org/TR/tenthold-1.

[13] W. Zhao, L. E. Moser, and P. M. Melliar-Smith. Fault tolerance for distributed and networked applications. *Encyclopedia of Information Science and Technology*, Idea Group, Inc., January 2005.