A Reservation-Based Extended Transaction Protocol for Coordination of Web Services

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

Firat Kart, L. E. Moser, P. M. Melliar-Smith
Dept. of Electrical and Computer Engineering
University of California, Santa Barbara
Santa Barbara, CA 93106
{fkart,moser,pmms}@ece.ucsb.edu

ABSTRACT:

Web Services can be used to automate business activities that span multiple enterprises over the Internet. Such business activities require a coordination protocol to reach consistent results among the participants in the business activity. In the current state-of-the-art, either classical distributed transactions or extended transactions with compensating transactions are used. However, classical distributed transactions lock data in the databases of different enterprises for unacceptable durations or involve repeated retries, and compensating transactions can lead to inconsistencies in the databases of the different enterprises. In this paper, we describe a novel Reservation Protocol that can be used to coordinate the tasks of a business activity. Instead of resorting to compensating transactions, the Reservation Protocol employs an explicit reservation phase and an explicit confirmation/cancellation phase. We show how our Reservation Protocol maps to the Web Services Coordination specification and describe our implementation of the Reservation Protocol. We compare the performance of the Reservation Protocol with that of the Two Phase Commit Protocol and Optimistic Two Phase Commit Protocol. We also compare the probability of inconsistency of the Reservation Protocol with that of Compensating Transactions.

KEY WORDS:
Business Activity, Continuous Availability, Extended Transaction Model, Relaxed Atomicity, Reservation Protocol, Transaction Processing, Web Services

INTRODUCTION

Business activities often involve related tasks that are carried out over a long period of time in a loosely-coupled distributed environment. Web Services (Champion, Ferris et al. 2002) make it possible to automate business activities across multiple 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, reliability, scalability and performance. Problems in the operation of the Web Services can adversely affect the relationships between an enterprise and its customers, suppliers and partners. Resolving inconsistencies among the databases of multiple enterprises is difficult, expensive, time-consuming and error-prone, much more so than within a database of a single enterprise.

Many enterprise applications are programmed using the transaction processing programming paradigm. A *transaction* is a set of operations on the application state that exhibit the Atomicity, Consistency, Isolation and Durability (ACID) properties (Gray and Reuter 1993).

For distributed transactions, commercial transaction processing systems have used the Two Phase Commit Protocol, which involves a coordinator and multiple participants (Gray and Reuter 1993). Although the Two Phase Commit Protocol might work well for coordination of operations within a single enterprise, the use of the Two Phase Commit Protocol in distributed transactions that span multiple enterprises unavoidably involves the locking of a data record of one enterprise by another enterprise. Even for successful transactions, data records are locked for significant periods of time, adversely affecting performance. If the transaction coordinator fails, this locking period might be too long for an enterprise to tolerate.

Optimistic Two Phase Commit Protocols (Herlihy 1986, Kung and Robinson 1981, Thomasian 1998) do not lock database records but rather search for conflicts between operations on database records when transactions are committed, with one or more transactions being aborted when a conflict is detected. In principle, Optimistic Two Phase Commit Protocols achieve the same degree of transaction isolation as the Two Phase Commit Protocol; even though the record is not locked physically, it is "locked" logically. Optimistic Two Phase Commit Protocols gamble that conflicts are rare; they achieve lower overheads when no conflicts occur at the expense of higher overheads when a conflict is detected.

Instead of implementing a business activity as a distribution transaction using the Two Phase Commit Protocol or an Optimistic Two Phase Commit Protocol, a business activity might be implemented as an extended transaction, where one or more localized transactions are executed and committed individually at each site, as in the sagas strategy (Garcia-Molina and Salem 1987). When a business activity must be rolled back, compensating transactions are applied to reverse the committed local transactions. Although useful in many cases, compensating transactions have their limitations. One problem is the possibility of cascading compensations that result from relaxation of the isolation property, *i.e.*, before the compensating transaction is applied, other transactions might see and depend on the results of the committed transaction and, thus, must also be compensated. Identifying such transactions is difficult, because there is no way to find them a priori. Furthermore, it might be difficult or impossible to compensate a committed transaction. 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 the compensation of the sales task, can result in inconsistencies. In general, the programming of compensating transactions is difficult and prone to error.

In this paper, we present a novel reservation-based extended transaction protocol that avoids the use of compensating transactions while achieving atomicity and consistency similar to or better than other existing extended transaction protocols. Each task within a business activity is executed as two steps. The first step involves an explicit reservation of resources according to the business logic. For example, if the task involves reserving two seats out of 200 available seats on an airline flight, those two seats are explicitly reserved in a separate step. In the interests of the airline, a fee that is proportional to the duration of the reservation can be associated with the reservation. The second step involves the confirmation or cancellation of the reservation. Each of these steps is executed as a separate traditional short-running transaction, as in the sagas strategy (Garcia-Molina and Salem 1987). However, because of the explicit reservation, other transactions cannot interfere with the business activity. Thus, degree 3 isolation (Gray and Reuter 1993) is achieved and, if a business activity must be abandoned, recovery involves only the cancellation of reservations for participants in that business activity.

The advantages of the Reservation Protocol over distributed transactions based on the Two Phase Commit Protocol or the Optimistic Two Phase Commit Protocol are:

- Database records are locked, both physically and logically, for only the duration of local transactions.
- Only the required amount of a resource is reserved, rather than locking the database record for the entire resource for an extended period of time.
- The Reservation Protocol provides better response times for the clients and better throughput and completion times at the server, particularly if there is contention among the clients for the same resources.

The advantages of the Reservation Protocol over compensating transactions are:

- The risk of inconsistency of the databases of different enterprises is reduced.
- The programming of reservations is easier than the programming of compensating transactions.

In this paper, we show how our extended transaction protocol can be implemented as a Reservation Protocol on top of the Web Services Coordination specification, in a similar manner to WS-AT and WS-BA. We describe our implementation of the Reservation Protocol as middleware libraries that are linked into the application processes at the client and the server. We compare the response time, throughput and completion time of our Reservation Protocol with that of the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol, based on experimental measurements of our implementations of the protocols. We also compare the probability of inconsistency of the Reservation Protocol with that of compensating transactions, based on a stochastic analytical model. That analysis shows that, when compensating transactions are used, the probability of inconsistency approaches unity as more business activities are executed. In the analytical model, the Reservation Protocol performs at least four orders of magnitude more business activities with lower risk of inconsistency than compensating transactions.

SYSTEM MODEL

A *business activity* is a unit of work that spans two or more enterprises and 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 business activity are partially ordered. Tasks that are not causally related can be executed concurrently, and causally related tasks are executed according to the partial order.

A task can be modeled as an operation on one or more resources, and typically modifies some of the attributes of those resources. For example, the task of purchasing/selling certain kinds of goods (application-defined 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.

Figure 1 shows an example business activity, highly simplified, of purchasing a product (resource) from a supplier and shipping that product. The purchaser proceeds with the purchase only if both the product and the shipping are available. The purchaser first places an order for the product and then arranges for shipping. However, if the shipper cannot deliver the product in time (*e.g.*, no trucks are available), the purchaser must reverse the previous purchase and apply a compensating transaction. Later in the paper, we show how our Reservation Protocol deals with this situation without the need for compensating transactions.

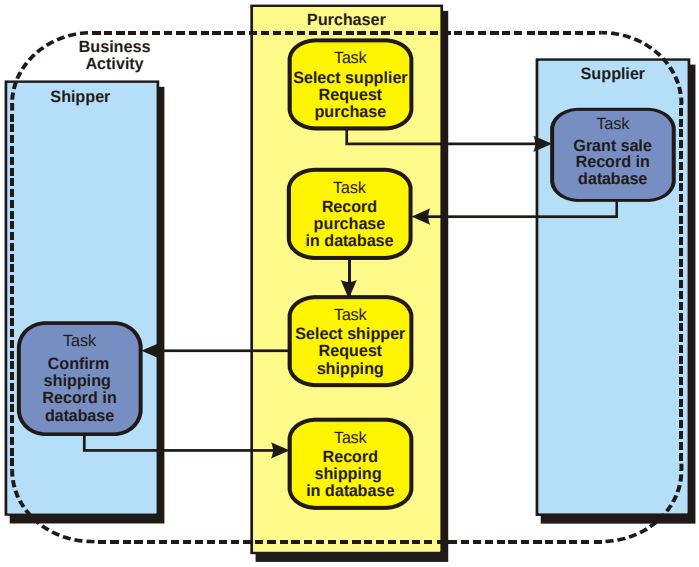

Figure 1. A business activity that comprises multiple tasks that span multiple enterprises. Each task is executed as a traditional transaction within one of the enterprises.

A business activity can be modeled as a *business transaction*, where there is a coordinator and multiple participants. In business transactions, the ACID properties of traditional transactions are not appropriate for the following reasons. First, it is not necessary that all of the participants see the same result. For example, if two suppliers are competing, one might see the reservation committed, while the other sees the reservation cancelled. Second, the failure of a task does not need to result in the rollback of the entire business transaction, particularly if there are multiple alternative suppliers. Third, the effect of executing a task might not be completely reversible, because of the business logic. For example, a reservation fee might still be imposed even if the business transaction is abandoned.

The coordinator and the participants in a business transaction (including the client and the server applications and/or middleware components) are subject to crash faults but not arbitrary (Byzantine) faults. In our Reservation Protocol, the coordinator for the business transaction is strongly coupled with the client application that initiated the business transaction. The coordinator runs in the same process as the client application. Therefore, the coordinator is regarded as a representative of the client, and the client is held accountable for any financial cost incurred if the client coordinator fails. However, the Reservation Protocol provides protection against failure of the client coordinator by a counterpart server coordinator, as discussed later in the implementation section.

The communication between the coordinator and the 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 the next section.

THE RESERVATION 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 traditional 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 supplier as a result of a request by the purchaser. 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 allows either outcome. However, the application is not free to use the resource until after the reservation is committed. Even if other client transactions see the intermediate result of the reservation, they see only resources that are available to them and not resources that have been reserved by other clients. All operations at the server, including for example augmenting the available resources, are undertaken as local transactions at the server.

In general, the reservation and commitment/cancellation steps imply that some form of service or work has been carried out. Consequently, an enterprise may impose a fee for the reservation and the associated service and work.

The coordination of different tasks within a business transaction is achieved in two phases, as shown in Figure 2 (where we use the same business activity as in Figure 1). In the first phase, the coordinator at the client 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. In the second phase, the client coordinator determines which reservations to confirm and which reservations to cancel, and then sends the confirmation/cancellation requests to the appropriate participants. The criteria, that determine which reservations to confirm and which to cancel, are application-dependent and, thus, out-of-scope of this paper.

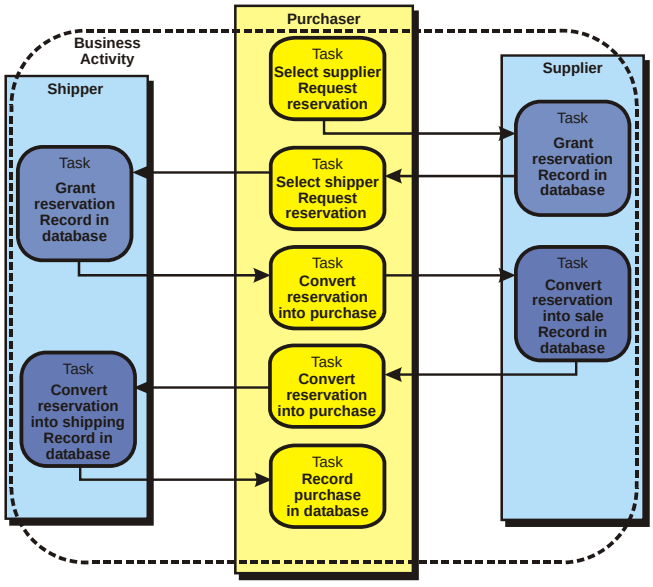

Figure 2. A business activity executed using the Reservation Protocol.

In a traditional distributed transaction based on the Two Phase Commit Protocol or the Optimistic Two Phase Commit Protocol, if a fault occurs, each participant can decide unilaterally whether to abort and rollback the transaction. In contrast, with our Reservation Protocol, only the client

coordinator is authorized to commit or cancel the transaction. A fault at other participants might affect the client's decision and, thus, the outcome of the transaction; however, it does not necessarily result in the rollback of the transaction.

Resource Reservation vs. Locking Resource vs. Resource Request Retry

At an abstract level, resource reservation in the Reservation Protocol, locking a resource in the Two Phase Commit Protocol, and resource request retry in the Optimistic Two Phase Commit Protocol are similar in that they all put the resource on hold temporarily. However, there are significant differences in the three protocols, as shown in Figure 3 and discussed below.

In the Reservation Protocol, the reservation of a resource and the confirmation or cancellation of the reservation, are executed as traditional ACID transactions. Between the reservation and the confirmation, the database records are unlocked and are available to other business activities. The application has control over the reservation, how much of the resource is reserved, and how long the resource is reserved.

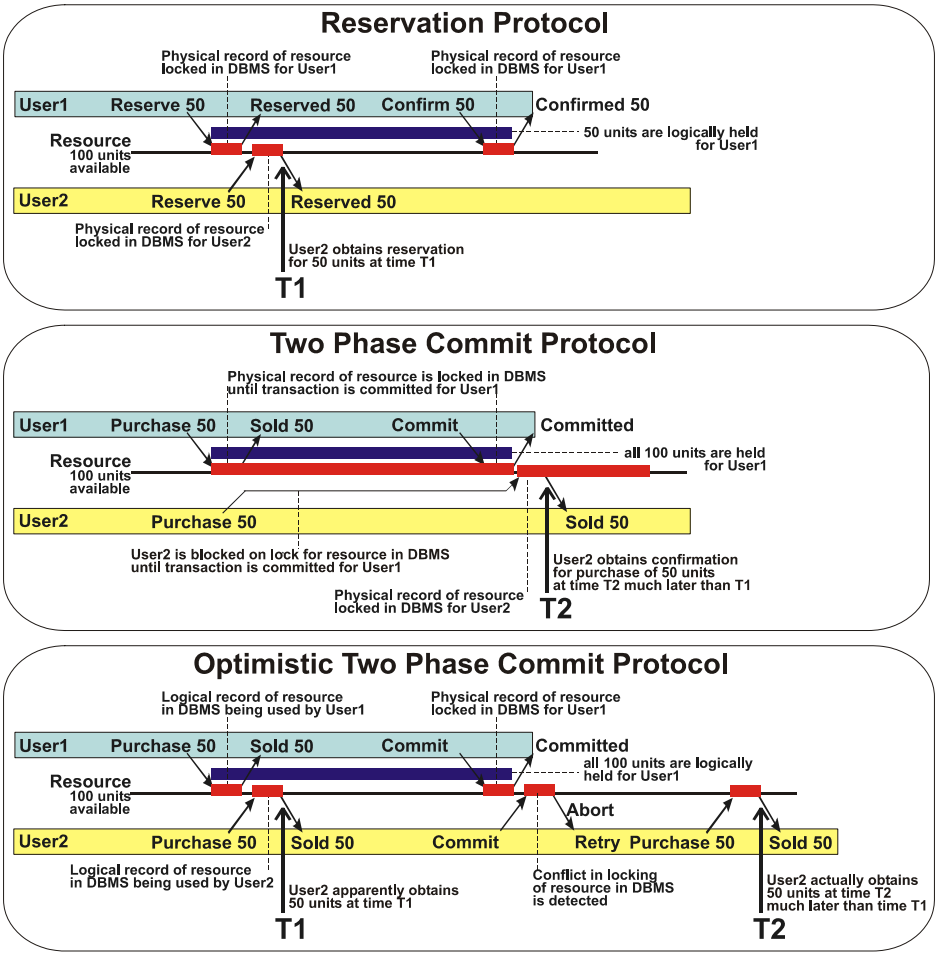

Figure 3. Comparison of resource reservation in the Reservation Protocol, locking in the Two Phase Commit Protocol, and request retry in the Optimistic Two Phase Commit Protocol.

In the Two Phase Commit Protocol with conservative concurrency control for database systems [Bernstein, Hadzilacos et al. 1987], 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 is locked. In practice, a timeout is used to control how long a transaction lasts and to prevent a resource from being locked too long. However, expiration of the timeout can result in inconsistency within or between the databases. The timeout used in a database system is quite distinct from the reservation period and timeouts used in our Reservation Protocol.

In the Optimistic Two Phase Commit Protocol (Herlihy 1986, Kung and Robinson 1981, Thomasian 1998), the database records are not physically locked during a transaction. Rather, when a transaction is committed, its database accesses are checked against those of other transactions. If a conflict exists, the conflicting transaction is aborted and the client of the failing transaction starts over. The Optimistic Two Phase Commit Protocol provides the same strong atomicity and isolation properties as the conservative Two Phase Commit Protocol, with lower overheads when no conflicts occur, at the price of higher overheads and longer delays when a conflict is detected.

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.,* although some of the resource has been reserved, a sufficient quantity remains available to satisfy the reservation). Thus, the application can take an appropriate action without delay. However, if the resource is locked by the database system and another transaction wants to access it, the new transaction must wait until the lock is released. The waiting time can be long, in which case the application cannot take immediate action.

Yet another difference between resource reservation and locking of resources is whether the owner of the resource can be paid for the resource reservation. In our protocol, the reservation of a resource forms a contract between the client and the resource owner which explicitly reflects the fee that the resource provider can charge. The fee for the reservation can be proportional to the duration of the reservation to discourage clients from reserving resources for excessively long periods of time. It is not obvious how to associate such a fee with the locking mechanism that is internal to the database system.

To understand better this benefit of resource reservation, consider the seats on an airline flight as a resource. A transaction might need to reserve, say, two seats on a flight while it checks other legs of the trip, the availability of a hotel room, etc. The database record is locked only briefly, for the time required to indicate that two seats are reserved, and then the lock is released. Another customer (transaction) might also need to reserve seats on the same flight, and it can do so as soon as the first transaction releases its briefly held lock. If the last seat on the flight has already been reserved, the transaction can acquire the lock for the flight, look at the seat record, and immediately inform the application that no seats are available. The application, or rather the customer, might then look for available seats on other flights. On the other hand, if the seat record is locked by the database system, 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, the Reservation Protocol ensures consistent results among the participants in a business transaction that have confirmed their tasks and, thus, it achieves a form of atomicity for the business transaction.

The behavior of the Reservation Protocol under fault conditions is shown in the flow chart in Figure 4. The first phase of the protocol consists of reservation actions. Once a participant (resource holder) has granted a reservation for a resource, it is committed to that reservation until the client (through the coordinator) explicitly cancels the reservation. For this to work in the business world, the participant may associate a fee with the reservation. To encourage a customer to decide quickly to confirm or cancel a reservation, the participant might use an exponentially increasing fee, *i.e.*, a long-duration reservation costs much more than a short-duration reservation.

In the second phase, if the client coordinator wishes to confirm or cancel a reservation but fails to communicate with a participant, all it must do is note the current time and then it can continue. Once a participant has granted a reservation for a resource, it is committed to honoring that reservation. The coordinator need not 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.

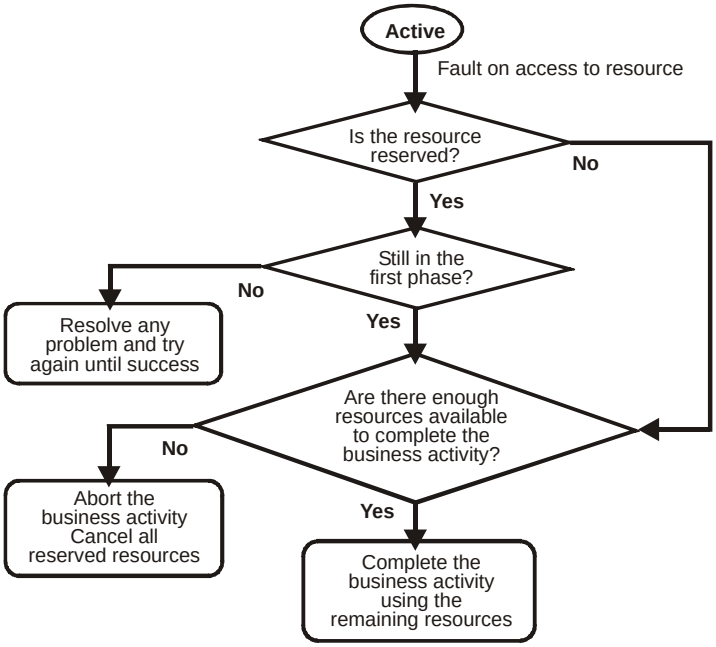

Figure 4. Fault handling flow chart for the Reservation Protocol.

If a fault occurs during the first phase, the effect is minor, *i.e.*, the fault limits the number of business partners from which the client coordinator can choose. If a critical business partner is not available and an alternative partner cannot be found, the coordinator stops the first phase and starts the second phase by canceling all existing reservations. In this case, the coordinator might still incur a fee for the reservations it made during the first phase, even though the entire business transaction has been rolled back. To avoid such a fee, 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.

Although we have assumed that a task within a business activity is a short-duration 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 specification (Cabrera, Copeland et al. 2004a, 2004b, 2004c).

Mapping to the Web Services Coordination Specification

The Web Services Coordination (WS-C) specification (Cabrera, Copeland et al. 2004a) describes an extensible framework that allows a variety of protocols for coordinating the interactions of distributed applications. The WS-C framework enables a service to create a context needed to propagate an action to other services and to register for coordination protocols.

The Reservation Protocol can be implemented on top of the WS-C framework in a manner similar to the business agreement protocols defined in the WS-BA specification (Cabrera, Copeland et al. 2004c). In this section, we define the Reservation Protocol in terms of states and protocol message types. The state diagram given in Figure 5 shows the state of an individual participant in a business activity, on receiving and sending protocol messages. The coordination logic among the participants is under the control of the application logic and, thus, is not defined at this level. Due to space constraints, we omit technical details related to the XML schemas, the related WSDL declarations, and the content of the business activity coordination context.

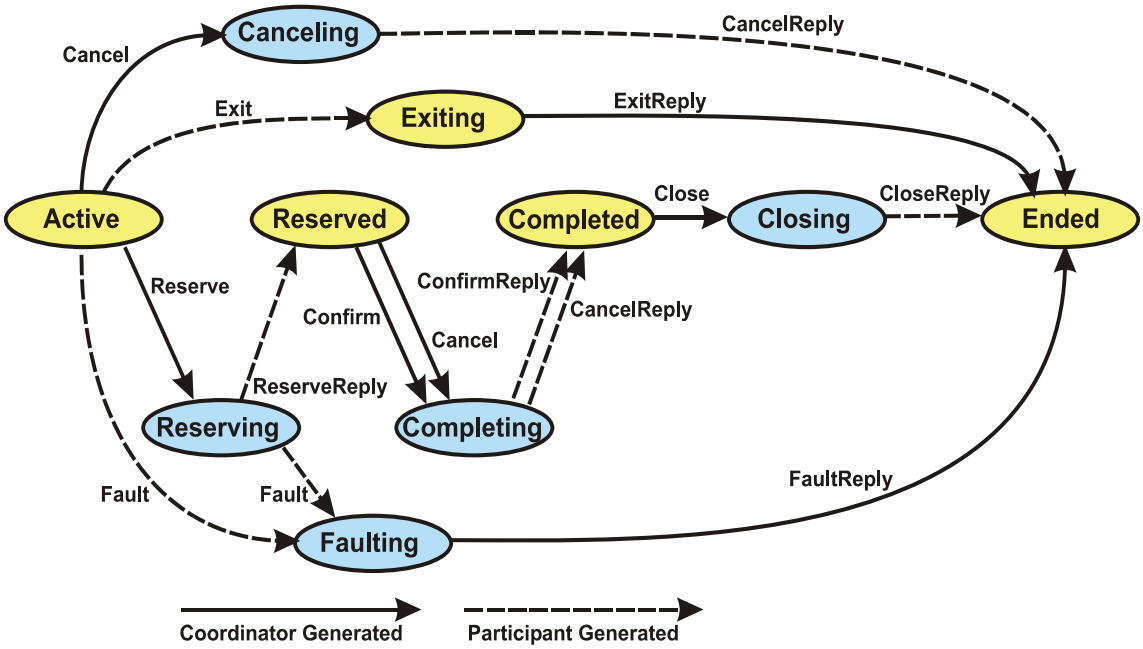

Figure 5. State diagram of the Reservation Protocol.

The coordinator sends the Cancel, Close, Forget, ExitReply, Reserve, Confirm and Cancel messages. The participant sends the Register, CancelReply, CloseReply, FaultReply, Exit, ReserveReply, ConfirmReply and CancelReply messages. The Reserve, ReserveReply, Confirm, ConfirmReply, Cancel and CancelReply messages carry application payloads that must be delivered to the appropriate applications. The message types and the states are described below.

- **Cancel/CancelReply** - The coordinator can remove a participant from a business activity while 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 Canceling state. The participant transitions to the Ended state when it sends a CancelReply message to the coordinator.

- **Exit/ExitReply** - 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 ExitReply message. The participant transitions to the Ended state when it receives the ExitReply message sent by the coordinator.

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

- **Reserve/ReserveReply** - The coordinator sends a Reserve message (with application payload to provide details about the reservation) when the client indicates that it wants to place a reservation with a participant. The Reserve message includes a reservation time (duration) by which the coordinator must send a Confirm/Cancel message after receiving the ReserveReply from the server. The coordinator must ensure that the reservation time is long enough, compared to the normal execution time for the business activity, so that the business activity semantics are not violated (*e.g.*, the coordinator sends a Confirm message for task A, assuming that there is enough time to confirm the reservation for task B but, by the time it sends the Confirm message for task B, the reservation for task B has timed out). On receiving the Reserve message, a participant transitions to the Reserving state and passes the message to the application so that the order can be filled. When the application indicates that the reservation has been completed, the participant sends a ReserveReply message to the coordinator. If the application declines the reservation request, a Fault message is generated and sent to the coordinator. On sending the ReserveReply message, the participant transitions to the Reserved state.

- **Confirm/ConfirmReply** - The coordinator sends a Confirm message to a participant when the client indicates that it wants to convert the reservation into a permanent order. On receiving the Confirm 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 ConfirmReply message to the coordinator and transitions to the Completed state.

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

- **Close/CloseReply** - When the two phases of the Reservation Protocol have 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 CloseReply message. The participant subsequently transitions to the Ended state.

Like the WS-BA protocols, the coordinator and the participants in the Reservation Protocol also accept the GetStatus and Status messages for query and response related to the current state. These operations are read-only and do not change the state. Furthermore, the Reservation Protocol requires that the coordinator and the participants are capable of handling duplicate messages.

Use Case

We now describe how to use the Reservation Protocol to coordinate a business activity. We consider a business activity in which a client wants to purchase a certain amount of a product from its suppliers and to arrange for shipment of the product. There are two suppliers for the product. Because the two suppliers are geographically far apart, the client must contact two different shipping companies, one for each supplier to provide shipping between that supplier and the client. To support the Reservation Protocol, the suppliers and the shippers offer the following Web Services operations: Reserve, Confirm and Cancel. In addition, we assume that they offer a RequestForQuote Web Service operation. First, we describe a scenario in which all of the resources are available and no fault occurs during the business activity.

As illustrated in Figure 6, at the beginning of the business transaction, the client (*i.e.*, the initiator of the transaction) asks the coordinator to create a new coordination context for the Reservation Protocol using the activation service provided by the coordinator (Step 1). The coordinator then creates a coordination context and returns it to the client (Steps 1 and 2). The coordination context contains an endpoint reference for the registration service so that the participants in the business activity can use the registration service to register for the Reservation Protocol. All messages exchanged between the client and each participant, and between the coordinator and each participant, include the coordination context created for the business activity.

During the business transaction, the client conducts a number of read-only tasks such as a RequestForQuote from each of the suppliers and each of the shipping companies (Step 3). Note that these steps are optional and that the client might start by going directly to the reservation phase. Because the RequestForQuote message contains the coordination context for the business transaction, the transaction participants register with the coordinator (Step 4). The Web Service responds to the RequestForQuote by sending the quotes to the client (Step 5). After having retrieved enough information from its business partners, the client asks the coordinator to start the reservation phase (Step 6).

The client coordinator then sends the Reserve messages to the participants (Step 7). Subsequently, the participants respond positively with ReserveReply message to the coordinator (Step 8). Then, the coordinator passes the reservation results to the client application for selection (Step 9). Next, the client directs the coordinator to start the second phase by confirming (Confirm) the reservations for supplier 1 and shipping company 1 and by canceling (Cancel) the reservations for supplier 2 and shipping company 2 (Step 10). The coordinator then does so (Steps 11-12 and Steps 15-16). As a result, the infrastructure at supplier 1 and the infrastructure at shipping company 1 send the ConfirmReply message to the client coordinator (Steps 13-14), and the infrastructure at supplier 2 and the infrastructure at shipping company 2 send the CancelReply message to the client coordinator (Steps 17-18). The coordinator then sends Close messages to all of the transaction participants (Step 19), and they respond by sending CloseReply messages to the coordinator (Step 20). The coordinator then informs the client that the business transaction has completed successfully (Step 21).

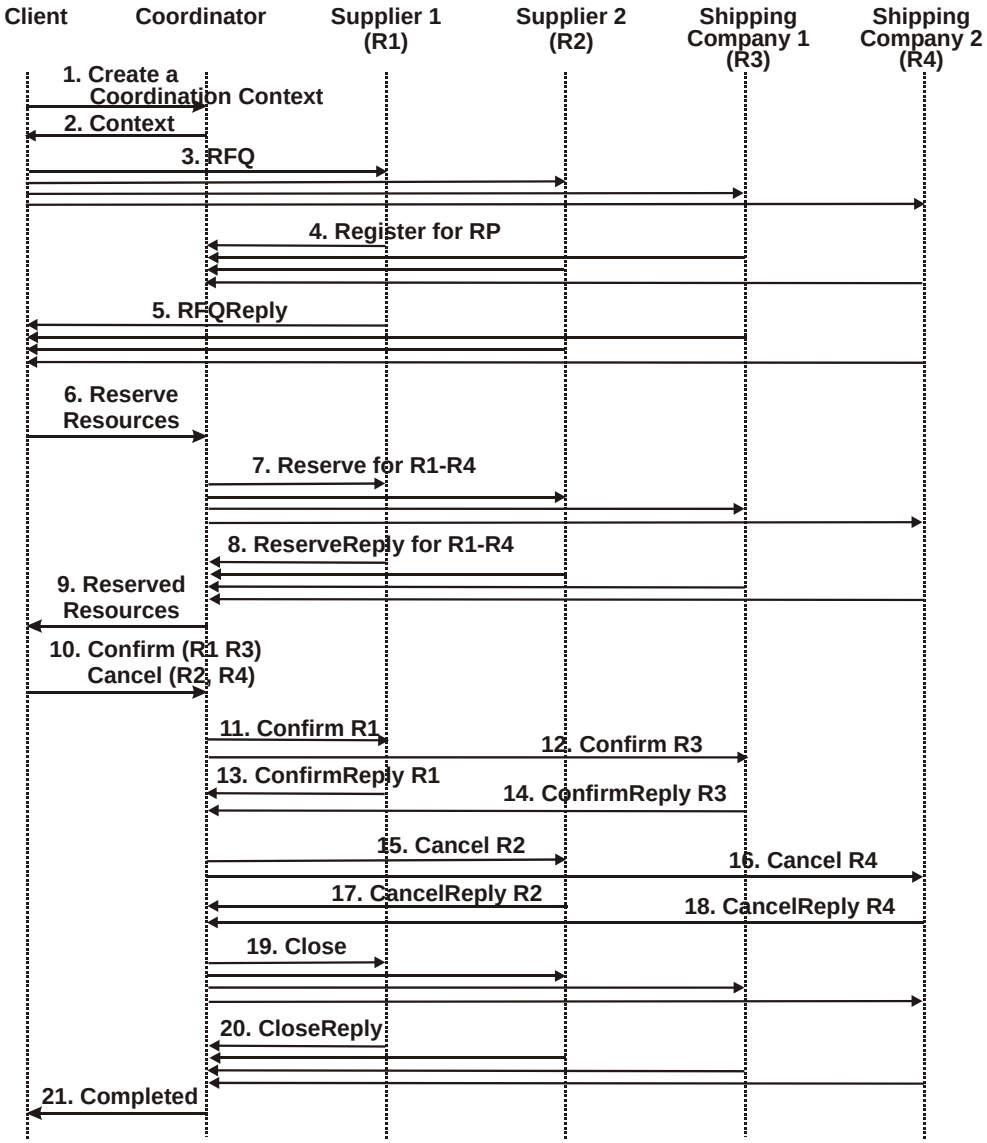

Figure 6. An example of a business transaction that uses the Reservation Protocol.

In another scenario shown in Figure 7, supplier 1 rejects the reservation request (by sending a Fault message) and supplier 2 accepts the reservation request. Thus, the client is forced to use shipping company 2. If shipping company 2 accepts the reservation request, the client confirms the reservation 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 (*e.g.*, 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 of the client, and might force the rollback of the business transaction.

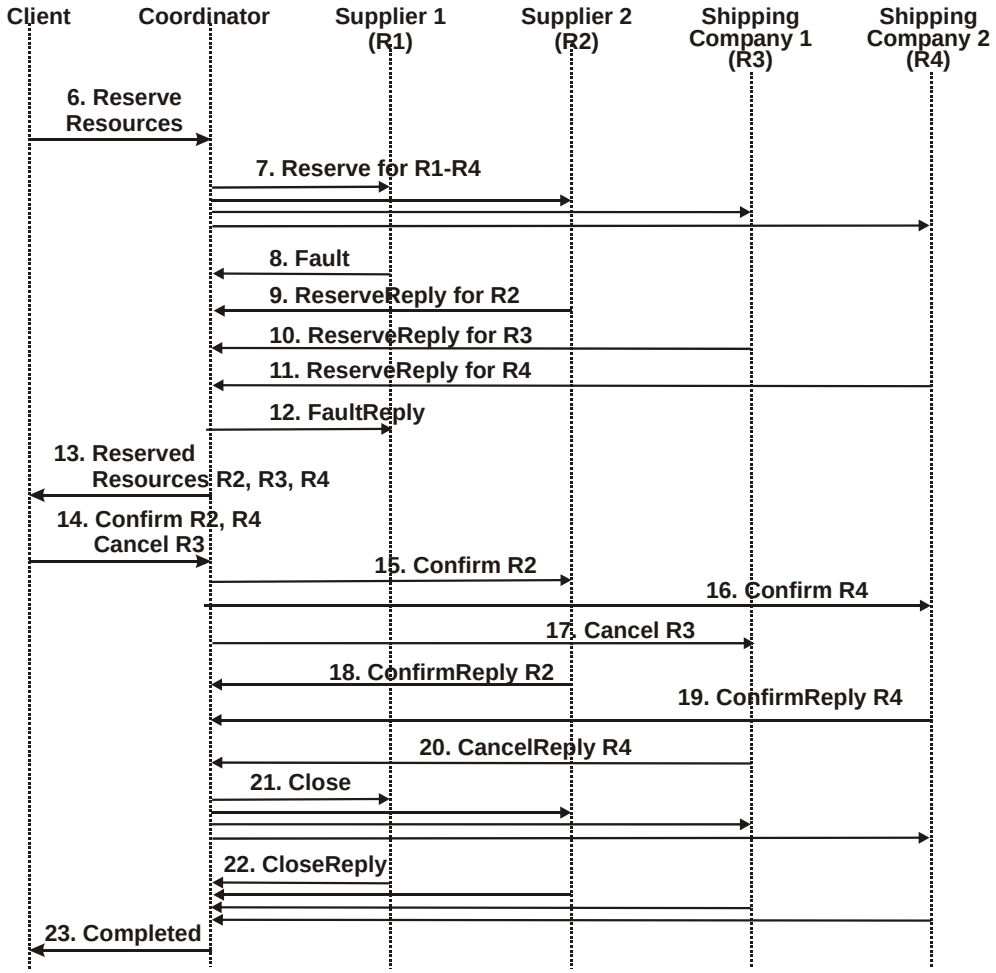

Figure 7. Another scenario of a business transaction that uses the Reservation Protocol.

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 the partner using different communication channels, *e.g.*, by telephone or fax, and/or can establish a dedicated communication channel with that partner.

In practice, reliable messaging (Bilorusets, Box et al. 2005) between the client coordinator and the Web Services providers must be used. Consider the following exceptional scenario. A Web Service (a participant in a business transaction) might crash immediately after it has accepted and handled a reservation request but before it sends the corresponding reply. As part of the recovery process, when the Web Service restarts, it looks for a record of the reply message and checks if the client has received it. If no such record is found, the Web Service has the option of canceling the reservation that it has made (without charging the client). Alternatively, it can proceed to regenerate the reply for the reservation request if the time that elapsed since the initial request is fairly short. The client coordinator must be ready to receive such reply messages out-of-sequence. If it has advanced to the second phase of the business transaction, the coordinator responds with a Cancel message. If it is still in the first phase, the coordinator has the option to accept the reservation even if it timed out the request earlier.

If a fault occurs after the participant has committed the reservation and the business transaction has advanced to the second phase, the coordinator cannot abandon the reservation. Moreover, the fault does 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. Instead, the ConfirmReply message or the CancelReply message contains an indication that an exception occurred. The coordinator invokes an exception handler registered by the client and the client makes a decision on how to proceed based on the business logic. For example, the client can cancel the reservation placed earlier with the faulty participant and confirm the reservation with another participant, if one is available. If such actions are not possible, the client corrects any problems on its side and 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, message logging and replay (Zhao, Moser et al. 2005a). The fee-based reservation policy also encourages an enterprise to ensure that its Web Services are highly available.

IMPLEMENTATION

We have implemented the Reservation Protocol as several Java packages. The implementation includes client and server middleware, with APIs that make it easy for the client application and the server application to use the Reservation Protocol. Figure 8 illustrates the overall architecture of our implementation of the Reservation Protocol, showing the client and server components of the middleware. The client and the server middleware are designed with modularity in mind, so that the client and server applications can be developed on top of these modules.

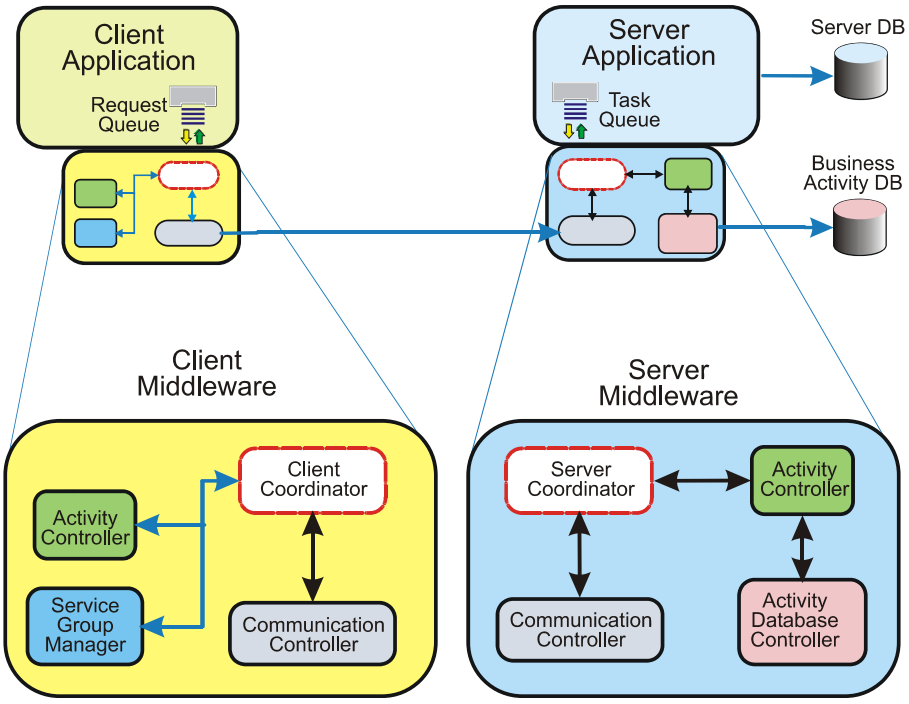

Figure 8. The Reservation Protocol architecture.

Client Middleware

The client middleware provides APIs for the client application and facilitates the location of Web Services. As shown in Figure 8, the client middleware consists of the following components:

- Client Coordinator
- Activity Controller
- Service Group Manager
- Communication Controller.

To fulfill the functionality of the middleware, the components of the client middleware interact with each other as shown in Figure 8 and perform the functions described below.

The *Client Coordinator* allows the client application to invoke API calls on it and coordinates the other components and information flow within the client middleware. The *Activity Controller* controls the lifecycle of the business activities of the client and informs the client application of activity information. The *Service Group Manager*, via the Communication Controller, obtains service information from the available servers and provides this information to the client application. The *Communication Controller* handles communication with its counterpart at the servers to which it is connected. For each server, there exists a separate handler for server communication. The communication between the client and the server is via SOAP messages.

Server Middleware

The server middleware provides APIs for the server application and keeps track of business activities and handles client connections. As shown in Figure 8, the server middleware consists of the following components:

- Server Coordinator
- Activity Controller
- Activity Database Controller
- Communication Controller.

To fulfill the functionality of the middleware, the components of the server middleware interact with each other as shown in Figure 8 and perform the functions described below.

The *Server Coordinator* invokes API callbacks on the server application, coordinates the other components and information flow within the server middleware and controls the message flow. The *Activity Controller* manages client business activities. When there is an update in a client activity, the Activity Controller informs the server application and the client middleware of that update. The Activity Controller keeps a timer for the reservation duration of the activity. The *Activity Database Controller* keeps information about the client activities and their status in the Business Activity Database. It provides the ability to recover and restart the server, if a fault occurs. The *Communication Controller* accepts client connections, handles communication with the client, and passes messages to the Server Coordinator.

Reservation Protocol Interactions

Figure 9 shows the Reservation Protocol interactions between the client and server applications and middleware.

When a client application wants to make a reservation with one or more Web Services, it makes a call to the client middleware, specifically to the Client Coordinator which communicates with the Service Group Manager. The Service Group Manager looks for available servers for each of the Web Services that the client requested and informs the Client Coordinator. There might be more than one server for each service type. The Client Coordinator then communicates with the Communication Controller, which sends the service requests to the respective servers and waits for replies from those servers.

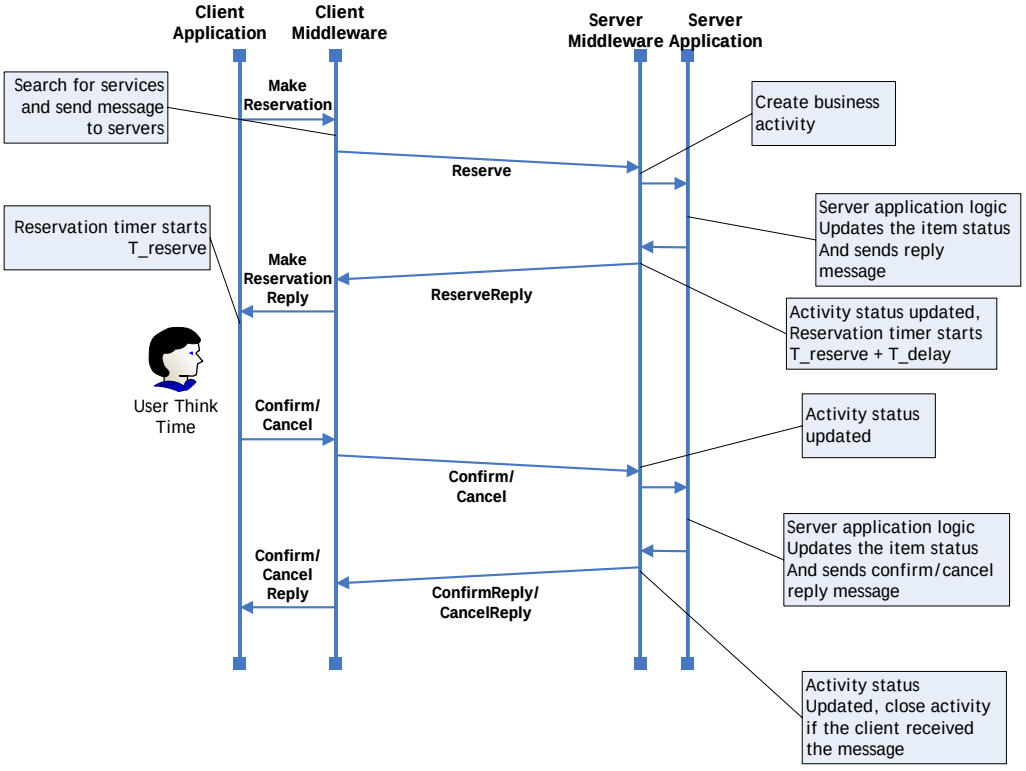

Figure 9: Reservation Protocol interactions.

When the server receives a Reserve message from the Communication Controller at the client, it informs the Server Coordinator, which interacts with the Activity Controller. The Activity Database Controller creates an entry in the Business Activity Database and then the Server Coordinator makes a call to the server application. The server application reserves its resources and replies, through a call to the Server Coordinator, which communicates with the Activity Database Controller and the Communication Controller. The Activity Database Controller updates the status of the activity in the Business Activity Database, and the Communication Controller sends a ReserveReply message to the respective client. At this point, the ActivityContoller starts the timer for the reservation, with the timeout value $T_{reserve} + T_{delay}$.

The duration T_{delay} covers the time for network delays, and is configurable. The duration $T_{reserve}$ is the amount of time that the client application has requested in its Reserve message to reply to the server, with the confirmed amount of reserved items, after it has received the ReserveReply message from the server. The Activity Controller at the server controls this time by maintaining a timer for each reservation. The client must send a Confirm/Cancel message to the respective server before the expiration of the timeout at that server's Activity Controller.

In our implementation, there is a coordinator component at both the client and the server, which work in concert with each other. The Client Coordinator decides and coordinates the lifecycle of the business activity. However, if the client fails or is unreachable (due to an application error, network failure, etc or because it is closed willingly), the Server Coordinator takes over control from the Client Coordinator. If the client has asked for a reservation time $T_{reserve}$, the Client Coordinator "thinks" the reservation time is $T_{reserve}$ and the Server Coordinator "thinks" the reservation time is $T_{reserve} + T_{delay.}$ Note that T_{delay} is a kind of lead time which ensures that the Server Coordinator does not take over control from the Client Coordinator prematurely if a fault occurs.

Note that the amount of the product that the client confirms must not exceed the amount that the client reserved, which is enforced by the client application. If the client application sends a Confirm message with a confirmed amount of 0, the value 0 indicates that the client does not want to purchase any of the product it reserved, in effect, it is canceling its reservation.

Note also that, when the client application makes a reservation request, it does not specify any particular server. The Service Group Manager is responsible for choosing an appropriate server to provide the service or product and for informing the Communication Controller at the client to make the necessary request calls. The Confirm message includes the server ID, which the Communication Controller uses to find the addresses of the relevant servers.

On receiving the Confirm message, the Activity Database Controller at the server updates the status of the reservation, unless it has already timed out. When a reservation times out, the Activity Database Controller at the server updates the Business Activity Database accordingly and informs both the client and the server applications about the cancellation through the Server Coordinator and the Communication Controller, respectively. If the client application sends the Confirm message just before it receives the cancellation notice from the server, the Activity Database Controller at the server is consulted before the Confirm message is sent.

Note that the server's ReserveReply message serves as an acknowledgment for the client's Reserve message and, similarly, the server's ConfirmReply/CancelReply message serves as an acknowledgment for the client's Confirm/Cancel message. Note also that the Communication Controller at the client does not send a message to the server, if a timeout has occurred.

PERFORMANCE EVALUATION

We have implemented the Reservation Protocol, the Two Phase Commit Protocol and an Optimistic Two Phase Commit Protocol, and have conducted measurements of the protocols based on our implementations. For the experiments, we have also written a driver program that generates, and controls, the clients' requests. Here we compare the performance of the protocols. The performance metrics that we use are the mean response time seen by the clients, the mean throughput at the server, and the mean completion time at the server. For the experiments, we used a simple business activity without dependencies or chains, because we are interested primarily in comparing the performance of the protocols. We conducted the experiments on 3.2 GHz Pentium 4 PCs with 2 GBytes of memory and a 100 Mbit/s local area network.

Request Distributions

We consider two different distributions of resource requests:

- Uniform random distribution
- Preferential distribution.

With the uniform random distribution, all resources have the same probability of being requested by a client. With the preferential distribution, some resources have higher probabilities than others of being requested by a client, which reflects the popularity of those resources and the clients' demand for those resources in the marketplace. For the preferential distribution, we consider three different resource request factors, 1.1, 1.5 and 2. The graph in Figure 10 shows the probabilities for resources being requested under the four distributions. For convenience, the graph shows 20 resources (instead of 100 resources which we used in our experiments). The horizontal axis represents the indices of the resources, and the vertical axis gives the probability of requesting a particular resource. For the preferential distribution with factor 2, for example, the first resource is requested with probability 0.5, the second resource is requested with probability 0.25, etc and, similarly, for the other preferential distributions.

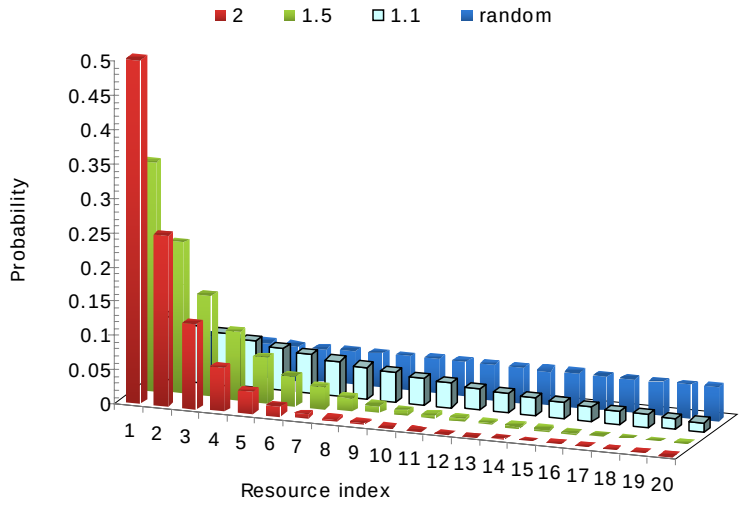

Figure 10: Resource request distributions.

Parameters for the Experiments

In our experiments there are 100 resources for which a client can make a request and the load on the server is increased by increasing the number of clients. In the experiments we set T_{delay} to 100 ms, where T_{delay} covers the time for network delays, as mentioned previously. The duration T_{next} is the duration of the interval between the client's completing a request and making its next request. In the experiments we set T_{next} = 60 seconds.

The "think" time T_{think} of the client is the duration of the interval between the client's receiving a ReserveReply message from the server, indicating that the server has reserved the amount of the resource requested by the client and the client's sending a Confirm/Cancel message to the server. In the experiments we set T_{think} = 100 ms, 1000 ms and 5000 ms.

Note that the client's think time does not affect the performance of the Reservation Protocol, but it is significant for the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol, as discussed below.

Reservation Protocol

Figure 11 shows the Reservation Protocol interactions and the times for those interactions. The client sends a Reserve message to the server, requesting a reservation. The server receives the Reserve message, processes the request, and informs the client about the availability of the resource it requested in a ReserveReply message. After receiving the client's Reserve message, in order to avoid database inconsistencies, the server must first claim a lock on the database record for the requested resource. If the database record is already locked, the server thread is blocked and must wait in a FIFO queue for the lock until the lock is released. When it obtains the lock, the server thread checks and updates the database record to reflect the reservation of the resource for that client, and then releases the lock. The Activity Database Controller at the server inserts an entry in the Business Activity Database to record an outstanding reservation, before it passes information to the server application. The server performs the same steps when a client sends a Confirm/Cancel message to the server.

For the Reservation Protocol, the response time of the client includes the network delay for two pairs of message exchanges, the lock request delay, the message processing time, and the database access time (two database record updates, four lock updates, and four activity record updates). The think time of the client does not affect the response time seen by the client.

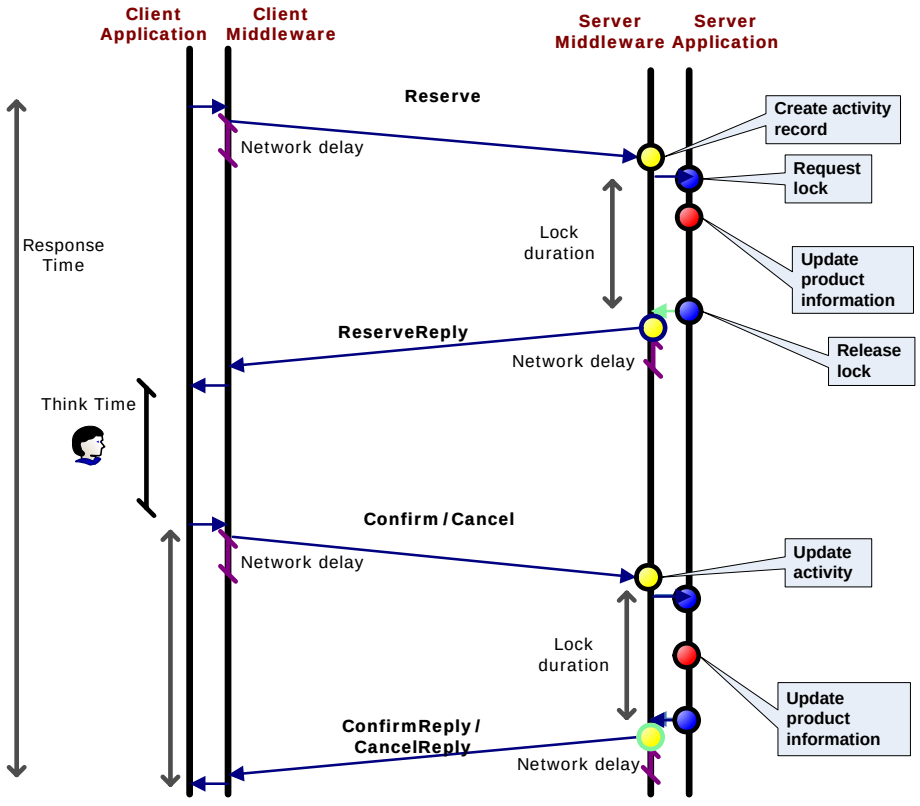

Figure 11: Times for the Reservation Protocol interactions.

Two Phase Commit Protocol

Figure 12 shows the Two Phase Commit Protocol interactions and the times for those interactions. When a client requests a resource, first the server gets the lock on the database record for the resource and then makes the necessary updates to the database record. After the client commits the transaction, the server releases the lock on the database record. Note that, for the Two Phase Commit Protocol, the server exchanges several additional messages with the client before it releases the lock on the database record.

For the Two Phase Commit Protocol, the response time seen by the client includes the network delay of three pairs of message exchanges, the lock request delay, the database access time (two database record updates and two lock updates), and the think time at the client.

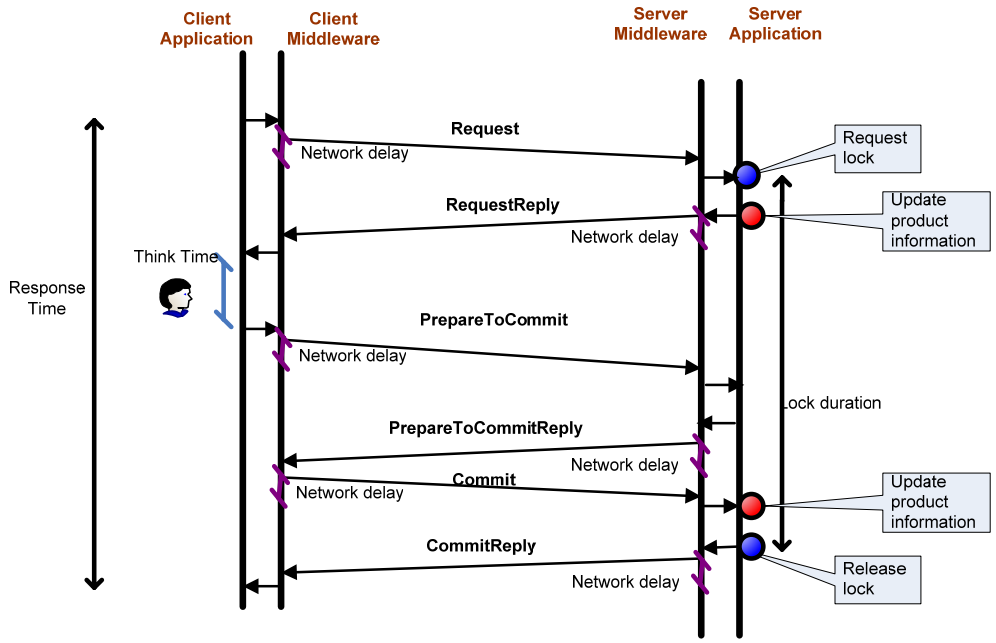

Figure 12: Times for the Two Phase Commit Protocol interactions.

Optimistic Two Phase Commit Protocol

Figure 13 shows the Optimistic Two Phase Commit Protocol interactions and the times for those interactions. When a client makes a request, the server gets a lock on the database record, obtains information from the record, releases the lock and returns the information to the client, typically with a timestamp. The data in the database record is not modified, and it is not locked.

When the client is ready to complete the transaction, the client requests the various servers involved to validate its prior requests. Each of the servers determines from the timestamps that the database record has not been updated by some other business activity, and returns a validate reply confirming the validity of the request. If all of the servers confirm the validity of the request, the client commits the transaction, as shown in Figure 13, and each server locks, updates, and unlocks its database record. If any server returns a negative validate reply, because the database record has been updated by some other transaction, the client aborts the current transaction and restarts it from the beginning, as shown in Figure 3.

The Optimistic Two Phase Commit Protocol is clearly similar to the Two Phase Commit Protocol, the difference being that, with the Optimistic Two Phase Commit Protocol, the database records are not held locked for long periods. If no conflict is detected during the validation phase, the Optimistic Two Phase Commit Protocol might have slightly lower overheads than the more conservative Two Phase Commit Protocol. If a conflict is detected and the transaction must be restarted, the Optimistic Two Phase Commit Protocol incurs higher overheads than the Two Phase Commit Protocol.

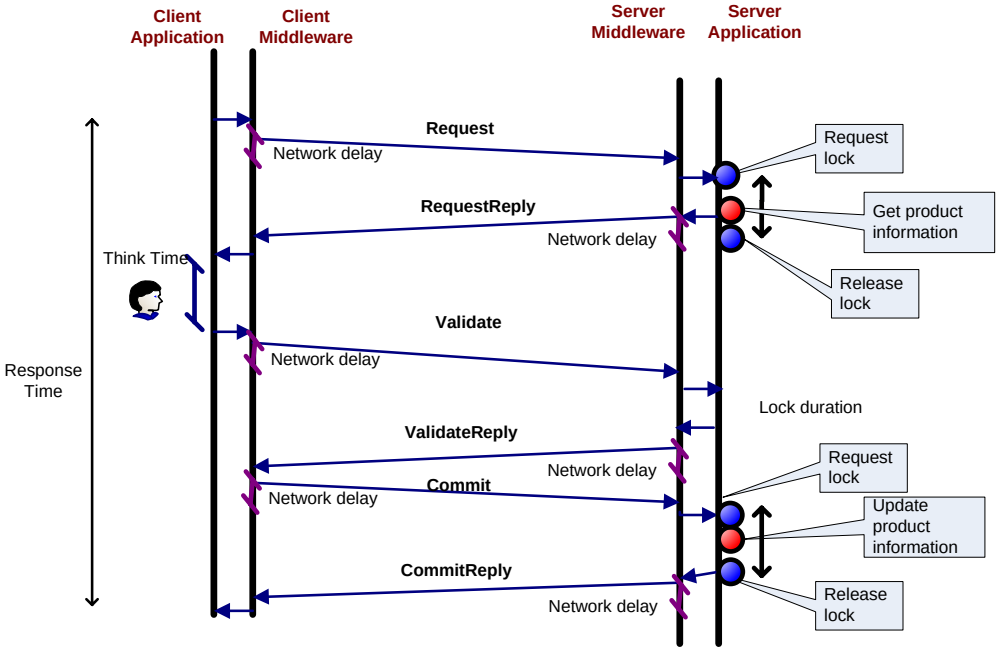

Figure 13: Times for the Optimistic Two Phase Commit Protocol interactions.

Response Time

As Figure 14 shows, under the uniform random distribution with think time 100 ms, the response times are almost constant as the number of concurrent requests (clients) increases. The response times of the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol are almost identical and about twice the response time of the Reservation Protocol. The reasons for the higher response times are lock contention for the Two Phase Commit Protocol and retries for the Optimistic Two Phase Commit Protocol and because more messages are exchanged which results in additional network delays.

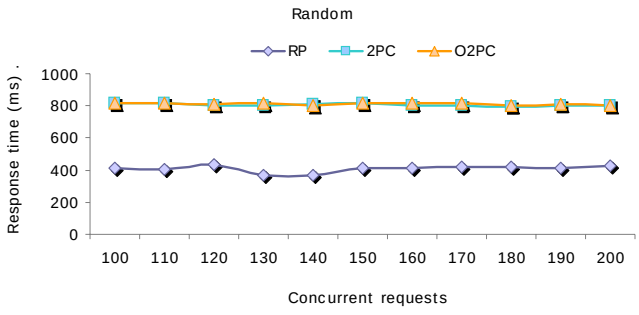

Figure 14: Response times for the uniform random distribution with think time 100 ms.

As Figure 15 shows, for the uniform random distribution with think time 1000 ms, the response time of the Reservation Protocol increases slightly as the number of concurrent requests (clients) increases, but it is still less than the response times of the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol, which are almost constant.

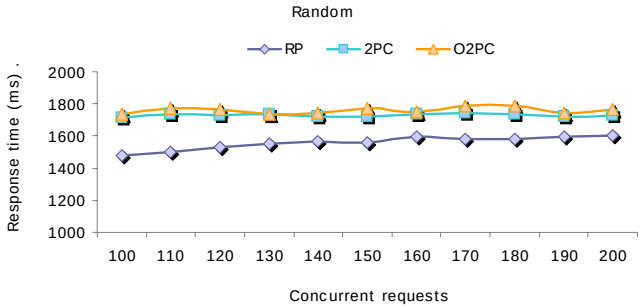

Figure 15: Response times for the uniform random distribution with think time 1000 ms.

As Figure 16 shows, for the uniform random distribution with think time 5000 ms, the response time of the Reservation Protocol increases slightly as the number of concurrent requests (clients) increases, but it is still less than the response times of the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol. The Optimistic Two Phase Commit Protocol has a higher response time due to increased retries of the transactions.

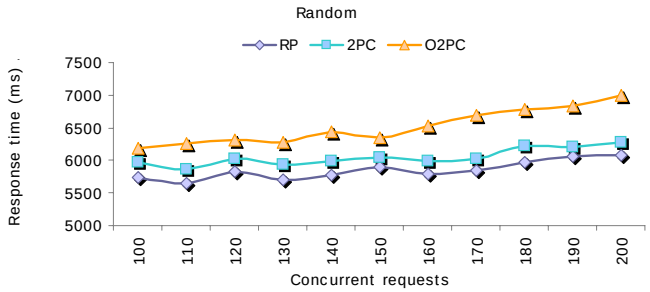

Figure 16: Response times for the uniform random distribution with think time 5000 ms.

Next, we investigated the response time seen by the clients for think time 100 ms under the preferential distribution with resource request factors of 1.1, 1.5 and 2.0.

As Figure 17 shows, for the preferential distribution with resource request factor 1.1 and think time 100 ms, the response time of the Reservation Protocol increases as the number of concurrent requests increases and almost approaches the response times of the other two protocols when the number of concurrent requests is 200. The response times of the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol are almost constant as the number of concurrent requests (clients) increase, but are greater than the corresponding response times of the Reservation Protocol.

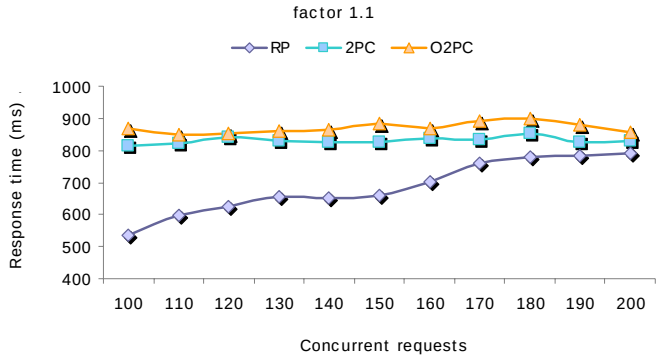

Figure 17: Response times for the preferential distribution with resource request factor 1.1 and think time 100 ms.

As Figure 18 shows, for the preferential distribution with resource request factor 1.5 and think time 100 ms, the response time of the Optimistic Two Phase Commit Protocol increases more than the response times of the other two protocols, as the number of concurrent requests increases. The response times of both the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol are greater than the response time of the Reservation Protocol.

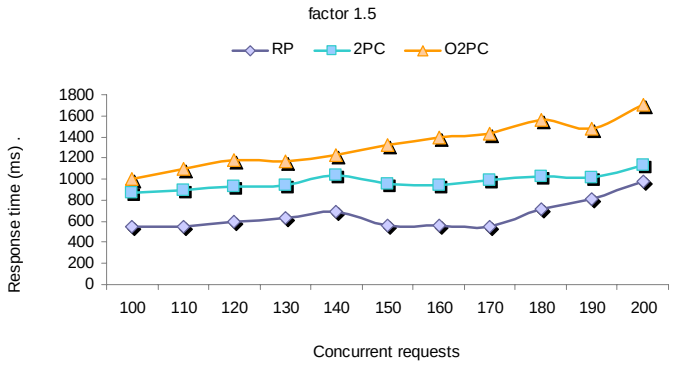

Figure 18: Response times for the preferential distribution with resource request factor 1.5 and think time 100 ms.

As Figure 19 shows, for the preferential distribution with resource request factor 2 and think time 100 ms, under increasing load, the response time of the Optimistic Two Phase Commit Protocol is considerably greater than that of the other two protocols.

For the preferential distribution with resource request factor 2, some of the resources (and the corresponding database records) have a much higher probability of being requested and, thus, the requests for those resources have a higher lock waiting time, before the server thread acquires the lock and continues processing the request. Thus, the Two Phase Commit Protocol has a higher response time than the Reservation Protocol, because there is more contention for the same resource, and a resource is locked while one of the clients is being served. The Optimistic Two Phase Commit Protocol has a higher response time than the Reservation Protocol, because there are many retries at the clients that did not acquire the resource in the validation phase. In the Optimistic Two Phase Commit Protocol, the client initiates the transaction and goes through the process until it succeeds. Although the client does not wait to acquire the lock for the resource as it does in the Two Phase Commit Protocol, it still needs to pass the validation phase successfully.

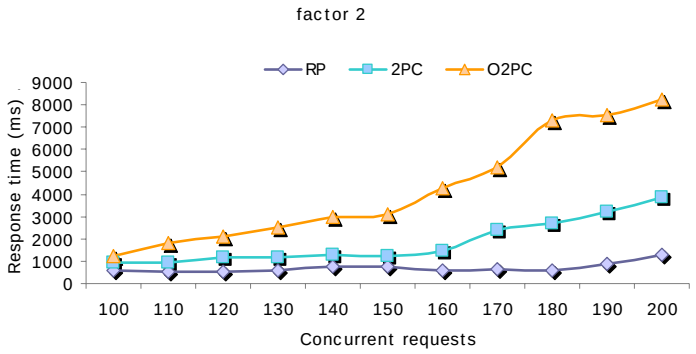

Figure 19: Response times for the preferential distribution with resource request factor 2 and think time 100 ms.

For a longer think time, the performance comparison of the protocols is even more striking, as the following graphs illustrate.

As Figure 20 shows, when the think time of the clients is increased to 1000 ms, the response times for both the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol increase substantially as the number of concurrent requests increases. The Two Phase Commit Protocol holds the resources longer, resulting in a longer response time. The Optimistic Two Phase Commit Protocol results in the clients going into the retry phase later if they are not able to pass the validation phase. The Reservation Protocol does not suffer from this increased think time because there are no locks on resources as in the Two Phase Commit Protocol, and no retries as in the Optimistic Two Phase Commit Protocol.

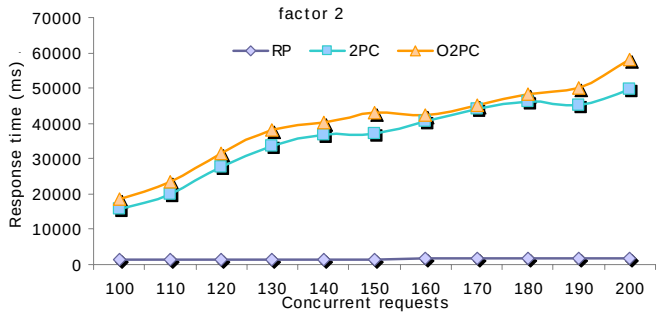

Figure 20: Response times for the preferential distribution with resource request factor 2 and think time 1000 ms. The response time of the Reservation Protocol ranges from 1433 ms to 1697 ms.

As Figure 21 shows, when the think time of the clients is increased to 5000 ms, the response time for the Reservation Protocol remains low but, for the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol, the response times are significantly higher. The response time of the Two Phase Commit Protocol reaches as high as 300 seconds and is worse than that of the Optimistic Two Phase Commit Protocol.

If the network delay is longer, the Optimistic Two Phase Commit Protocol is affected more than the Two Phase Commit Protocol. The reason is that, once the Two Phase Commit Protocol sends a message to the server, there are no retries and each retry of the Optimistic Two Phase Commit Protocol is affected by the longer network delay.

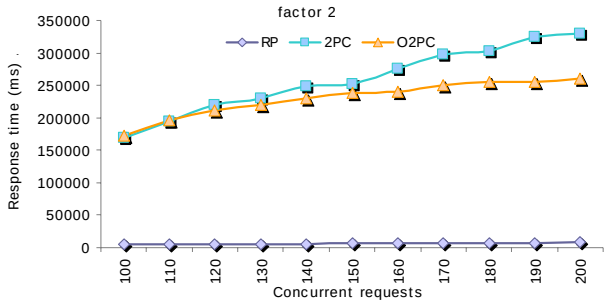

Figure 21: Response times for the preferential distribution with resource request factor 2 and think time 5000 ms. The response time of the Reservation Protocol ranges from 5897 ms to 8317 ms.

Throughput

We also investigated the throughput at the server for different think times of the clients and the random and preferential distributions with different resource selection factors. The throughput curves are most interesting when the clients contend for the resources; thus, we present the throughput curves only for the preferential distribution with resource request factor 2.

As Figure 22 shows, with think time 100 ms, as the number of concurrent requests increases, the throughput of all three protocols increases, without much difference between them.

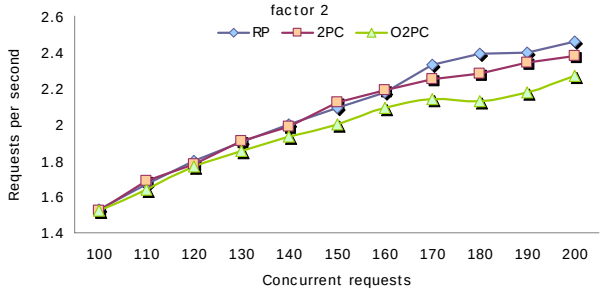

Figure 22: Throughputs for the preferential distribution with resource request factor 2 and think time 100 ms.

As Figure 23 shows, with think time 1000 ms, the throughput of the Reservation Protocol continues to increase, whereas the throughputs of the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol are almost constant.

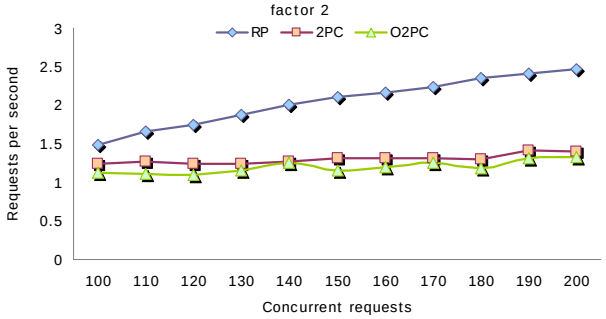

Figure 23: Throughputs for the preferential distribution with resource request factor 2 and think time 1000 ms.

As Figure 24 shows, with think time 5000 ms, the difference in throughput of the Reservation Protocol and the throughputs of the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol is even more striking. Again, the throughput of the Reservation Protocol continues to increase as the number of concurrent requests increases, but the throughputs of the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol remain almost constant. Again, the Two Phase Commit Protocol suffers from resource locking and the Optimistic Two Phase Commit Protocol suffers from retries, whereas the Reservation Protocol is not so affected.

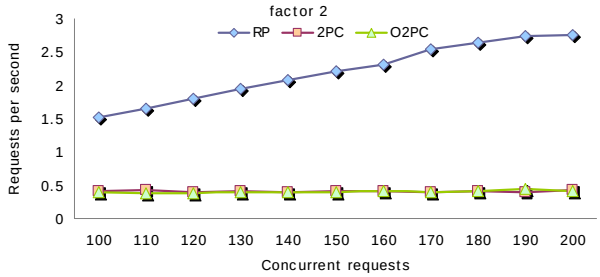

Figure 24: Throughputs for the preferential distribution with resource request factor 2 and think time 5000 ms.

Completion Time

We have also investigated the completion time at the server as the number of concurrent requests increases, for different think times. In this experiment, there is one resource at the server and all of the clients request the same resource. When the server receives the first request, we start a timer at the server. Once the server has completed the requests for all of the clients, we stop the timer and determine the time it took the server to complete those requests.

As Figure 25 shows, for think time 100 ms, the completion time for the Reservation Protocol is better than that for the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol. At 200 concurrent requests, the contention for the resource is high, but the Reservation Protocol is able to complete the requests in about 1/3 the time that it takes the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol.

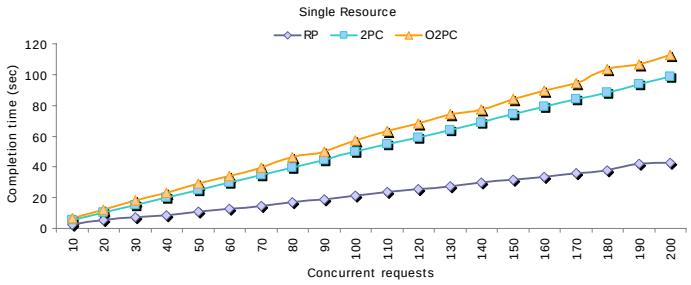

Figure 25: Completion times with a single resource and think time 100 ms.

As Figure 26 shows, for think time 1000 ms, the completion time of the Reservation Protocol is almost 10 times better than the completion times of the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol.

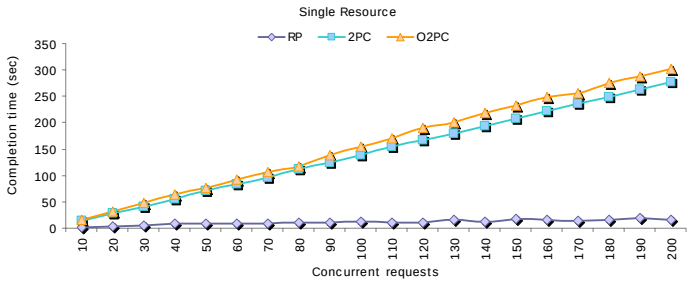

Figure 26: Completion times with a single resource and think time 1000 ms.

As Figure 27 shows, for think time 5000 ms, the ratio of the completion times of the Reservation Protocol to the Two Phase Commit Protocol and the Reservation Protocol to the Optimistic Two Phase Commit Protocol is about 40.

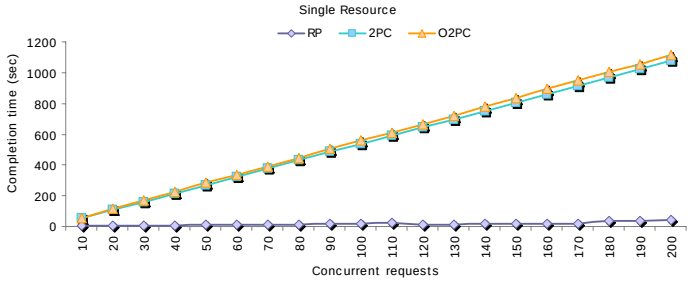

Figure 27: Completion times with a single resource and think time 5000 ms.

Considering all three metrics (response time, throughput, completion time), the Reservation Protocol performs better than the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol, particularly if the think time of the clients is long. The reason is that, when the think time is long and the clients contend for the same resource, the performance of the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol are adversely affected by the locking of resources and repeated retries, respectively. The Reservation Protocol does not suffer from such adverse effects.

Web Services, and business activities that span multiple enterprises, have relatively long think times, ranging from seconds to minutes, particularly if humans are involved. Our experiments indicate that the performance of the Reservation Protocol will be better than the performance of the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol for Web Services and business activities that span multiple enterprises.

Probability of Inconsistency

The probability that the databases within the same or different enterprises are left in an inconsistent state is a critical performance metric. We have constructed Markov models to represent the probability of inconsistency for compensating transactions and the Reservation Protocol. The results are shown below.

Figure 28 shows the probabilities of inconsistency for compensating transactions with 20 local transactions (tasks) per business activity. The essential weakness of the compensating transactions strategy, which results in the high risk of inconsistency, is that a fault in a local task causes the entire business activity to be aborted, with the effect that each local task must be

reversed by a compensating transaction (because nothing is known about the nature of the local tasks) before the entire business activity can be retried. The consequence is a large number of compensating transactions. A fault in a compensating transaction necessarily causes a risk of database inconsistency.

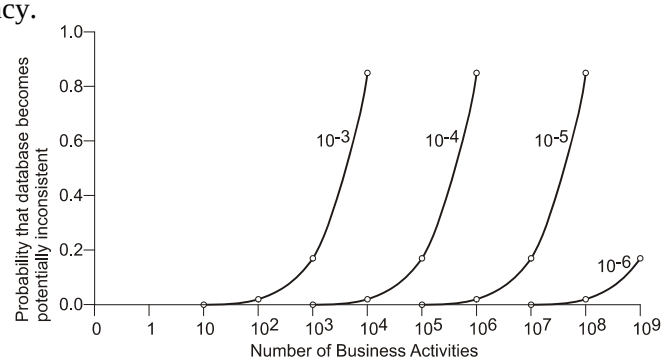

Figure 28: For compensating transactions, the probability that the databases are left in a potentially inconsistent state as the number of business activities increases.

Figure 29 shows the probability of inconsistency for the Reservation Protocol with 40 local transactions (20 tasks each with a reservation step and a cancellation/confirmation step) per business activity. The Reservation Protocol has less risk of database consistency than compensating transactions because a fault in a local task can be recovered by aborting and retrying the local task and, thus, there are fewer additional local transactions for each fault recovery, resulting in a lower probability of database inconsistency. This advantage of the Reservation Protocol arises, because the reservation and confirmation/cancellation steps are constrained and well understood. It is our assessment that the difference in the probability that the database is left in a potentially inconsistent state presents a decisive advantage for the Reservation Protocol.

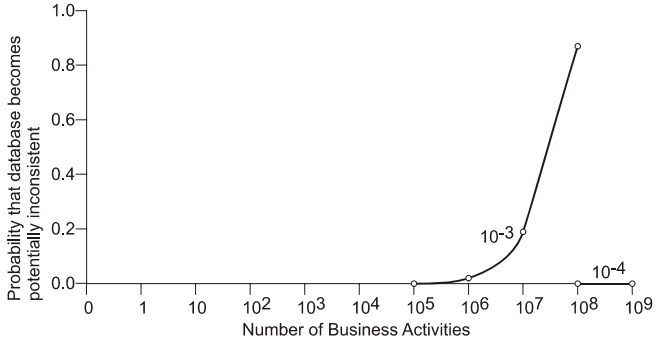

Figure 29: For the Reservation Protocol, the probability that the databases are left in a potentially inconsistent state as the number of business activities increases.

.

RELATED WORK

Broadly speaking, our Reservation Protocol is related to the extended transaction model and associated concurrency control mechanisms (Bernstein, Hadzilacos et al. 1987, Gray and Reuter 1993). The concept of reservation has been widely used in practical systems that support business services, such as hotel, airline and car rental reservations.

The business agreement protocols defined in the Web Services Business Activity Framework (Cabrera, Copeland et al. 2004c) are flexible enough to support many types of business activities. In particular, the BusinessAgreementWithCoordinatorCompletion protocol with the Mixed

Outcome coordination logic is compatible with our protocol. However, the protocols in that framework are not explicitly two-phase, and they depend on the use of compensating transactions to handle exceptions. These properties create some issues that must be circumvented to apply our Reservation Protocol in the Web Services Business Activity Framework.

The most closely related work to our Reservation Protocol is the Business Transaction Protocol (BTP) of the Organization for the Advancement of Structured Information Standards (OASIS 2002), 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, the BTP specification mentions only a reservation-like action as one of several ways to provide provisional or tentative state changes. The BTP specification does not pursue the concept of reservation to the extent that we do, and 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 (Ginis and Chandy 2000). In their protocol, a consumer makes timed-reservation requests to the service providers in the form of purchasing options for the use of resources provided by the service providers. In the second phase, the confirmation/cancellation requests are sent to the service providers. Compensating transactions are used to wipe out the effect of a partially fulfilled plan and to 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. Their atomic reservation protocol is not necessarily appropriate for use in loosely-coupled, long-running distributed business activities for which our Reservation Protocol is specifically designed.

The sagas strategy (Garcia-Molina and Salem 1987) allows a long-running transaction to be executed as a number of ACID transactions. Sagas can see intermediate results of other sagas; they rely on compensating transactions to handle exceptions; and they provide relaxed ACID properties for long-running transactions. The ConTract model (Wachter and Reuter 1992) is intended for defining and controlling complex long-lived activities at a level above ACID transactions. It includes the properties of persistency, consistency, recovery, synchronization, cooperation, and the use of assertions as invariants on entry to and exit from activities.

The Tentative Holding Protocol (THP) (Roberts and Srinivasan 2001, Roberts, Collier 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. THP allows tentative non-blocking holds (reservations to be requested) for a business resource. The primary benefit of THP is that it minimizes the possibility of rolling back committed transactions by providing more accurate information on the availability of the business resource to the client. 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 taken the resource. In such a case, business rules require the rollback of the business activity, and the client must apply a compensating 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 avoid the need for compensating transactions.

Several researchers (Alonso, Agrawal et al. 1996, Kamath and Ramamritham 1998, Worah and Sheth 1997) have considered extended transaction models in the context of workflow management systems. In many ways the issues that arise in loosely-coupled distributed systems are more complex variations of the issues that arise in workflow management systems.

Other researchers (Fekete, Greenfield et al. 2003, Greenfield, Fekete et al. 2003a and 2003b) have been engaged in a project that focuses on maintaining consistency in loosely-coupled distributed environments. They have proposed a research agenda based on an infrastructure that includes a language to express consistency conditions, tools to check whether the system maintains consistency, and guidance in using the infrastructure properly.

The Two Phase Commit Protocol (Gray and Reuter 1993) for atomic commitment of distributed transactions and our Reservation Protocol both 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, the protocols differ in that the Reservation Protocol requires the holding of a resource as a traditional atomic transaction, which does not prevent other transactions from being executed. To compensate the blocking hold of the resource, the Reservation Protocol allows the resource provider to require the client to pay a fee, based on the duration of the reservation. Furthermore, the 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. As shown by our performance measurements, the Reservation Protocol performs better than the Two Phase Commit Protocol, in terms of response time seen by the client, throughput of the server, and completion time of the server because the Reservation Protocol does not require the locking of resources.

The Optimistic Two Phase Commit Protocol (Kung and Robinson 1981, Herlihy 1986, Thomasian 1998) optimistically assumes that conflicts for resources will not occur. It requires a client to retry if another client already holds a lock on the resource, rather than waiting for that client to release the lock. As shown by our performance measurements, the Reservation Protocol performs better than the Optimistic Two Phase Commit Protocol, in terms of response time seen by the client, throughput of the server, and completion time of the server because the Reservation Protocol does not involve repeated retries.

CONCLUSION AND FUTURE WORK

Business activities rely on extended transaction protocols to coordinate individual tasks in order to achieve a coherent result. In this paper, we have described a novel reservation-based extended transaction protocol that can be used to coordinate business activities. Instead of resorting to compensating transactions, our Reservation Protocol employs an explicit reservation phase as a means of providing for later changes.

The Reservation 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. The Reservation Protocol is similar to the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol to some extent, *i.e.*, they are all two-phase and involve exclusive resource holdings, physically for Two Phase Commit and logically for Optimistic Two Phase Commit. However, there are many significant differences that make the Reservation

Protocol more suitable for coordinating business activities based on Web Services in a loosely-coupled distributed environment. As our experimental measurements have shown, the performance of the Reservation Protocol, in terms of the mean response time seen by the client, the throughput at the server, and the completion time at the server, is better than that of the Two Phase Commit Protocol and the Optimistic Two Phase Commit Protocol. Moreover, our analytical models show that the Reservation Protocol has a lower risk of database inconsistency than compensating transactions.

We plan to carry out future work along three directions. First, we plan to refine our 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 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. The Reservation Protocol favors a resource-centric approach. Integrating the protocol into a comprehensive Service Oriented Architecture is an interesting and challenging project. Third, we plan to extend our implementation and evaluation of the Reservation Protocol so that it handles dependencies and nesting among the tasks of a business activity.

ACKNOWLEGMENT

This research was partially supported by MURI/AFOSR Contract F49620-00-1-0330 and by UC Discovery Grant/QAD COM05-10194 for the authors at the University of California, Santa Barbara, and by a faculty startup award for the author at Cleveland State University.

REFERENCES

Alonso, G., Agrawal, D., El Abbadi, A., Kamath, M., Gunthor, R., Mohan, C. (1996). Advanced transaction models in workflow contexts, *Proceedings of the 12th International Conference on Data Engineering*, New Orleans, LA, February-March 1996, 574-581.

Bernstein, P. A., Hadzilacos, V., Goodman, N. (1987). *Concurrency Control and Recovery in Database Systems*, Addison Wesley, Reading, MA.

Bilorusets, R., Box, D., Cabrera, L. F., Davis, D., Ferguson, D., Ferris, C., Freund, T., Hondo, M. A., Ibbotson, J., Jin L., Kaler, C., Langworthy, D., Lewis, A., Limprecht, R., Lucco, S., Mullen, D., Nadalin, A., Nottingham, M., Orchard, D., Roots, J., Samdarshi, S., Schewchuk, J., Storey, T. (2005). Web Services Reliable Messaging Protocol. Retrieved November 22, 2006 from http://www-128.ibm.com/developerworks/library/specification/ws-rm/#main .

Cabrera, L. F., Copeland, G., Feingold, M., Freund, T., Johnson, J., Kaler, C., Klein, J., Langworthy, D., Nadalin, A., Orchard, D., Robinson, I., Shewchuk, J., Storey, T. (2004a). Web Services Coordination. Retrieved November 22, 2006 from http://www.ibm.com/developerworks/library/ws-coor/ .

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. (2004b). Web Services Atomic Transaction. Retrieved November 22, 2006 from http://www.ibm.com/developerworks/library/ws-transpec/ .

Cabrera, L. F., Copeland, G., Freund, T., Klein, J., Langworthy, D., Leymann, F., Robinson, I., Storey, T., Thatte, T. (2004c). Web Services Business Activity Framework. Retrieved November 22, 2006 from http://www.ibm.com/developerworks/ library/ws-busact/ .

Champion, M., Ferris, C., Newcomer, E., Orchard, D. (2002). Web Services Architecture. Retrieved November 22, 2006 from http://www.w3c.org/TR/2002/WD-ws-arch-20021114/ .

Fekete, A., Greenfield, P., Kuo, D., Jang, J. (2003). Transactions in loosely coupled distributed systems, *Proceedings of the 14th Australasian Database Conference*, Adelaide, Australia, February 2003, 7-12.

Garcia-Molina, H., Salem, K. (1987). Sagas, *Proceedings of the SIGMOD Conference*, San Francisco, CA, 1987, 249-259.

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

Gray, J., Reuter, A. (1993). *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Publishers, Inc., San Francisco, CA.

Greenfield, P., Fekete, A., Jang, J., Kuo, D. (2003a). What are the consistency requirements for B2B systems?, *Proceedings of the High Performance Transaction Systems Workshop*, Asilomar, CA, October 2003.

Greenfield, P., Fekete, A., Jang, J., Kuo, D. (2003b). Compensation is not enough, *Proceedings of the 7th IEEE International Enterprise Distributed Object Computing Conference*, Brisbane, Australia, September 2003, 232-239.

Herlihy, M. (1986). Optimistic concurrency control for abstract data types, *Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing*, Calgary, Alberta, Canada, August 1986, 206-217.

Kamath, M., Ramamritham, K. (1998). Failure handling and coordinated execution of concurrent workflows, *Proceedings of the IEEE International Conference on Data Engineering*, Orlando, FL, February 1998, 334-341.

Kung, H. T., Robinson, J. T. (1981). On optimistic methods for concurrency control, *ACM Transactions on Database Systems*, 6(2), 213-226.

Limthanmaphon, B., Zhang, Y. (2004). Web Service composition transaction management, *Proceedings of the 15th Australasian Database Conference, Conferences in Research and Practice in Information Technology*, 27, Dunedin, New Zealand, January 2004, 171-179.

Organization for the Advancement of Structured Information Standards (OASIS). (2002). Business Transaction Protocol, version 1.0. Retrieved November 22, 2006 from http://www.oasis-open.org/committees/businesstransactions/ .

Roberts, J., Collier, T., Malu, P., Srinivasan, K. (2001). Tentative hold protocol part 2: Technical specification. Retrieved November 22, 2006 from http://www.w3.org/TR/tenthold-2 .

Roberts, J., Srinivasan, K. (2001). Tentative hold protocol part 1: White paper. Retrieved November 22, 2006 from http://www.w3.org/TR/tenthold-1 .

Thomasian, A. (1998). Concurrency control: Methods, performance, and analysis, *ACM Computing Surveys*, 30(1), 1998, 70-119.

Wachter, H., Reuter, A. (1992). The ConTract model, *Database Transaction Models for Advanced Applications*, ed. A. K. Elmagarmid, Morgan Kaufmann Publishers, Inc., San Francisco, CA, 219-263.

Worah, D., Sheth, A. (1997). Transactions in transactional workflows, *Advanced Transaction Models and Architectures*, eds., S. Jajodia and L. Kershberg, 1997, 3-34.

Zhao, W., Moser, L. E., Melliar-Smith, P. M. (2005a). Fault tolerance for distributed and networked applications, *Encyclopedia of Information Science and Technology*, 2005, Idea Group Publishing, Hershey, PA, 1190-1196.

Zhao, W., Moser, L. E., Melliar-Smith, P. M. (2005b). A reservation-based coordination protocol for Web Services, *Proceedings of the IEEE International Conference on Web Services*, Orlando, FL, July 2005, 49-56.

ABOUT THE AUTHORS

Wenbing Zhao received the Ph.D. degree in Electrical and Computer Engineering from the University of California, Santa Barbara, in 2002. Currently, he is an assistant professor in the Department of Electrical and Computer Engineering at Cleveland State University. His current research interests include distributed systems, computer networks, fault tolerance and security. In the past, he has conducted research in the fields of superconducting materials and quantum optics. Dr. Zhao has more than 40 academic publications.

Firat Kart is a candidate for the Ph.D. degree in Electrical and Computer Engineering at the University of California, Santa Barbara, where he has served as a teaching assistant and a graduate student researcher. He received the B.S. degree in Computer Science from Bilkent University in Ankara, Turkey. His research interests include distributed systems, computer networks, Service Oriented Architectures, Web Services, database transactions, and supply chain and healthcare applications.

Louise E. Moser is a professor in the Department of Electrical and Computer Engineering at the University of California, Santa Barbara. Her research interests span the fields of computer networks, distributed systems and software engineering. Dr. Moser has authored or coauthored more than 230 conference and journal publications. She has served as an associate editor for the IEEE Transactions on Computers and an area editor for IEEE Computer magazine in the area of networks, and has also served on many conference program committees. She received a Ph.D. in Mathematics from the University of Wisconsin, Madison.

P. M. Melliar-Smith is a professor in the Department of Electrical and Computer Engineering at the University of California, Santa Barbara. Previously, he worked as a research scientist at SRI International in Menlo Park. His research interests encompass the fields of distributed systems and applications, and network architectures and protocols. He has published more than 250 conference and journal papers in computer science and engineering. Dr. Melliar-Smith is a pioneer in the field of fault-tolerant distributed computing. He received a Ph.D. in Computer Science from the University of Cambridge, England.