PER-REQUEST CONTRACTS FOR WEB SERVICES TRANSACTIONS

David Paul, Frans Henskens and Michael Hannaford
Distributed Computing Research Group, School of Electrical Engineering and Computer Science
University of Newcastle, Callaghan NSW, Australia

Keywords: ACID reduction, Semantic atomicity, Tentative hold.

Abstract: To allow providers to keep their autonomy and ensure the overall system can run satisfactorily, it is common practice in the Web Services environment for providers to reduce the strength of some of the traditionally-required ACID properties when offering transactional support. However, current standards require providers to offer a constant level of transactional support for each operation they provide. We describe a method that allows service providers to dynamically decide on the level of transactional support to offer for each client request. This allows the provider to base the level of transactional support offered on the current state of the system and internal logic, resulting in potential benefits for both service providers and consumers.

1 INTRODUCTION

Web Services transactions combine services provided by multiple, possibly completely unrelated parties into a single action. While these transactions are loosely based on the techniques used in conventional databases, strictly enforcing ACID properties in the Web Services environment is often impractical. Thus, it is typical to reduce the strength of some of the traditional transaction properties when combining multiple service calls into a transaction-like workflow. The currently supported reductions are static; they must be decided before the transaction begins, and all parties must provide the same level of support, even if some would be willing to support stronger versions of the properties. This paper presents some concepts and evaluation leading to the development of a protocol that allows the dynamic specification of the strength of various ACID properties for Web Services transactions.

The ACID properties of Atomicity, Consistency, Isolation, and Durability are well understood in traditional databases (Gray and Reuter, 1993). However, these are typically single systems, where a transaction coordinator has complete control over every aspect of the system. In the Web Services environment, transactions involve multiple autonomous entities working together, and very few are willing to give up the autonomy that ensuring the full ACID properties would require. Further, as Web Services transactions can run for long periods of time (perhaps days or weeks), complete support for the ACID properties can severely negatively affect the performance of both individual systems and the system as a whole.

Current techniques (e.g. (Ceponkus et al., 2002; Cox et al., 2004; Bunting et al., 2003)) reduce these problems by relaxing some of the ACID properties. However each entity in the transaction must provide the same level of transactional support. We believe that allowing different entities to dynamically provide different reductions to the ACID properties will improve support for transaction-like workflows in the Web Services environment. In our model, different providers in a transaction can choose the level of support for transactions that they desire. The client can then combine actions with different transactional guarantees into a cohesive whole that takes full advantage of the transactional support offered by each service. A simulator has been created to evaluate such a system.

Using this simulator it is possible to test how different reductions to the ACID properties work in different situations. The model can also facilitate changes to parameters such as the network topology, failure rates of particular services, or the number and length of transactions being run in such a way that allows repeatable testing with changes only to the transactional properties being used. The simulator will

Paul D., Henskens F. and Hannaford M.
PER-REQUEST CONTRACTS FOR WEB SERVICES TRANSACTIONS.
DOI: 10.5220/0002801800810088
In *Proceedings of the 6th International Conference on Web Information Systems and Technology (WEBIST 2010)*, page
ISBN: 978-989-674-025-2

also be used to develop and test a dynamic protocol that allows the strength of the various ACID properties supported to be negotiated during run-time.

This paper is organised as follows: Section 2 gives background information about transactions in the Web Services environment, and points out useful research for the study discussed in this paper. Section 3 then describes a motivating scenario to demonstrate the usefulness of per-request contracts between clients and service providers. This scenario is examined in Section 4, and the results discussed in Section 5. Finally Section 6 concludes the paper and indicates future directions.

2 BACKGROUND AND RELATED WORK

There are two approaches to the specification of Web Services transactions. The first, used in protocols such as the Transaction Internet Protocol (Lyon et al., 1998), is to attempt to move traditional transactions to the Web Services environment. The second is to redefine the concept of a transaction. Rather than specifying that the ACID properties must all be met, protocols such as BTP (Ceponkus et al., 2002), WS-Transaction (Cabrera et al., 2005; Cox et al., 2004) and WS-CAF (Bunting et al., 2003) reduce some of the ACID properties to allow transaction-like workflows that are better suited to the Web Services environment.

The Transaction Internet Protocol (Lyon et al., 1998) is based on the concept of two-phase-commit, which has a transaction coordinator that first asks each participant in the transaction to guarantee that it can perform the requested action. Each participant then sends a response to the coordinator specifying whether or not they will give that guarantee. If each participant does agree, then the coordinator sends a confirmation message to each of them; the participants have all guaranteed that the actions will complete successfully, so the transaction will also be successful. If one or more participants cannot give the requested guarantee, then the coordinator must let all participants know that the transaction has failed.

Once a participant has guaranteed that resources are available, it must ensure that these resources are reserved until the coordinator sends its second message. This can take an indeterminate amount of time, and possibly even forever if there is a break in a communication link. During this time, the participant must keep the requested resources available for the client, which can negatively affect other client interactions with that system.

Each service provider in the Web Services environment is autonomous, so they are typically not willing to give up the control necessary to support full ACID transactions through a two-phase commit protocol. Instead they typically offer a reduced level of transaction support. The most common reduction for Web Services transactions is to replace the ACID property of atomicity with semantic atomicity (Garcia-Molina, 1983). Semantic atomicity allows any requested action to be completed immediately (if possible), and, if the transaction has to roll back, a compensating action is performed to undo the already completed action.

Semantic atomicity typically removes the property of isolation from Web Services transactions, which can result in the overall system behaving in an undesirable manner. For example, a request may be denied because a currently running transaction has already been allocated the requested resource. If that transaction later fails and the compensating action returns the resource, then, in retrospect, the denied request could have been approved. While there are techniques that allow full support of isolation (Alrifai et al., 2006; Choi et al., 2005), this support necessarily affects how well services can perform.

There are different techniques that attempt to reduce the problems involved with ignoring global isolation while still providing an acceptable level of service. One such method is the Tentative Hold Protocol (Roberts and Srinivasan, 2001), which replaces the exclusive locks required by other techniques with non-exclusive ones. In this protocol, a client requests some resources from a provider, and the provider sends a reply that informs the client whether the requested resources are currently available. If the resources are available then the client can continue, assuming that the resources will be available when they are ready to commit. If ever the situation changes, and the requested resources become unavailable, then the service provider can notify the client and the transaction can be cancelled.

It is also possible to combine multiple reductions of transactional properties to allow the benefits of each technique to be used. For example, (Limthanmaphon and Zhang, 2004) allows support of both tentative holds and, through the use of compensating actions, semantic atomicity. (Fauvet et al., 2005) also includes two-phase commit. These combinations are useful, but the proposals only allow the use of the reductions explicitly built into them; service providers are forced into supporting a subset of the allowed reductions, which may not include the level of support they wish to provide.

While most standards specify support for only

certain transaction-like behaviours and reductions, there has been some work into offering more general support. For example, transactional attitudes (Mikalsen et al., 2002) allow providers to specify their transaction-like behaviour using an extension to their WSDL service descriptions. This allows the provider to explicitly describe how its actions can be used in a transaction, and allows for the combination of multiple different levels of transaction support, using middleware to hide the transactional complexities this requires from the client. However, providers are restricted to offering only one level of transaction support for each published service; it is impossible for the provider to change the level of support from one request to another.

This project believes that there is a number of possible reductions to the ACID properties that result in a useful transaction-like workflow. In particular, we believe that allowing different reductions to be used by different providers in a single transaction can improve performance of both transactions and individual providers. Further, we believe that a provider should be able to dynamically decide the level of transactional support to provide for a particular request based on internal logic, its current state, and the incoming request. For the purpose of this paper, we look at services that can be provided with either semantic atomicity or tentative holds, as specified in the next section.

3 EXAMPLE SCENARIO

Consider three providers offering a competing service that clients wish to use in a transaction. This service allows a client to order a certain number of finite resources (for example, this could be a booking of a number of hotel rooms, or an order for a number of books). Each provider is free to choose the level of transaction support that it wishes to make available, and each service is identical except for the level of transaction support being offered.

The providers choose between the following three levels of transaction support: (It is assumed that clients can query the level of transactional support offered by a service provider at any time.)

- Support for Semantic Atomicity. A client sends a request asking the provider for a certain number of resources. If the provider offers these to the client, then those resources cannot be offered to any other client. The client can, however, later cancel the order without penalty, in which case the resources again become available for later clients.
- Support for Tentative Holds. A client can send a request asking if the provider could fulfil a particular order. The provider sends back a message indicating whether the requested number of resources are currently available. Whilst the client has not finalised the booking, the provider is able to use the requested resources for other clients. If the resources become unavailable before the client finalises its booking, the provider lets the client know that the hold is no longer valid. Once a client has finalised an order, it cannot be cancelled without penalty.
- Variable Support. The provider offers semantic atomicity while resources are plentiful, but switches to only supporting tentative holds when the number of resources it can provide reach a certain level.

The behaviour of clients depends on the level of transactional support offered by the providers. Some clients will refuse to use any provider that does not support semantic atomicity. As the service offered by these providers is only one part of the client's transaction, using a service that only offers a tentative hold may lead to some actions in a transaction completing successfully before another action fails. To ensure that the client has no risk of penalty for a failed transaction, it may be necessary for the client to undo a booking without penalty, which is only possible if semantic atomicity is offered.

Other clients will not care about the level of transaction support offered for this service. These clients are either using the order for this service as a pivot action (Zhang et al., 1994) (a single action in a transaction such that its success or failure determines the success or failure of the entire transaction), or are willing to accept the risk that either this order succeeds while their other actions fail, or that this order fails while their other actions succeed.

Finally, some clients will prefer to use a provider that offers semantic atomicity, but, if necessary, will be willing to use one that only provides tentative holds. These clients are similar to those that do not care about the level of transaction support being offered, but are only willing to risk having this booking complete and the other actions fail if no better alternative exists.

In any case, each client finds a provider that offers a level of transaction support that they are happy with. If no such provider is found then the transaction fails with no action being taken. Otherwise, if this provider offers semantic atomicity, the client books the resources and then attempts the other actions in their transaction. If the other actions fail then the client cancels the order and the transaction fails without penalty. If the other actions succeed, then the transaction has completed successfully.

If, on the other hand, the found provider only offers a tentative hold to the client, then the client's action depends on the risk they are willing to take. If the client is willing to have this service succeed with a possibility that the other actions in the transaction fail, then they complete this order immediately, and then attempt the other actions. Successful completion of the other actions means a successful completion of the entire transaction. Failure of the other actions, however, means that the transaction fails and the client cannot cancel its completed order with the provider without a penalty.

The final case is where the found provider offers a tentative hold, and the client is willing to have the other actions complete successfully while the order for this service fails. Once the client has the tentative hold on the resources, it attempts to complete the other actions. If these fail then the client releases its tentative hold and the transaction fails without penalty. However, if the other actions complete successfully then the client attempts to finalise the order. As long as the tentative hold is still valid then this will be successful, resulting in a successful transaction. However, it is possible for the tentative hold to become invalid while waiting for the other actions to complete. If this occurs, the client can attempt to find the resources from another provider, but if this is impossible the transaction will fail with the other actions having been completed successfully. If the other actions are not undoable, this will result in a partially completed transaction, which may mean a penalty for the client.

To ease the burden on the client, the above logic can be moved into an abstract service (Schäfer et al., 2007). Each of the three competing providers is registered with the abstract service, and clients then send their requests to this service rather than to any of the providers individually. The abstract service then takes the responsibility for finding an appropriate provider for any booking or tentative hold request. In particular, in the event that the abstract service has a tentative hold that is later cancelled, the abstract service can attempt to find the resources from another provider, only informing the client of any problem if no other provider can satisfy the request.

4 EXAMINING THE SCENARIO

In order to determine whether having providers that dynamically change their transactional support for a service is beneficial, a software model of the situation in Section 3 has been created. This model simulates the sending of messages between clients and service providers, monitoring each message that is sent. The clients directly interact with the three service providers, though the logic would be similar if each client instead utilised an abstract service for communicating with the different providers.

In the model, the transactions of 1000 clients are generated to be started between times 1 and 100, with the start time of each transaction chosen randomly. It is assumed that the majority of these clients (80%) would prefer providers that offer semantic atomicity over those only offering tentative holds, with the other 20% being equally split between those unwilling to use any provider that does not offer semantic atomicity and those who do not care about the transaction support offered by the providers. Given that each provider offers an otherwise identical service, it is fair to assume that the majority would prefer the provider that offers a better transactional guarantee.

For the first test, as well as their other activities, each client wishes to order (randomly) between 1 and 10 resources from one of the three providers. The second test is identical except half of the clients wish to order a large number of resources (50). When a client needs to choose which provider to use, it randomly chooses between any of the providers that offer the required resources with the requested transactional guarantee.

Of the clients willing to risk having a partially completed transaction (that is, all except those who insist on semantic atomicity), half are assumed to risk having this service complete and the other actions in their transaction fail, and the other half to risk having this service fail while the other actions in their transaction complete. The results of failures with penalties would be skewed differently if these rates were changed, but, for this study, all that is important is the fact that a failure with penalty occurred, not whether that penalty was on this service or the other actions in the transaction.

Each client also wishes to perform some other actions as a part of their transaction. These other actions are modelled as an activity that takes between 1 and 10 time units, and fails 20% of the time. In reality, the other actions could consist of multiple activities, but, for this study, it is sufficient to model them as a single activity, since all that is important is whether they succeed or fail.

The three providers are tested using each of the three possible transactional guarantees. When using the variable scheme that changes from semantic atomicity to tentative holds, the provider changes to use tentative hold if the request would bring the number of available resources to under 50% of the initial resources offered by the provider. This is sufficient for

Table 1: Results when providers have limited resources.

Provider 1 Protocol	Provider 2 Protocol	Provider 3 Protocol	Clients (%)				
				Failure with			
			Success	No penalty	Penalty on other activity	Penalty on this service	Any penalty
Tentative	Tentative	Tentative	61.2	28.7	7.6	2.5	10.1
Variable	Tentative	Tentative	61.3	29.9	6.3	2.5	8.8
Variable	Variable	Tentative	62.2	29.5	5.7	2.6	8.3
Variable	Variable	Variable	61.6	31.7	3.6	3.1	6.7
Semantic	Tentative	Tentative	61.9	31.6	4.6	1.9	6.5
Semantic	Variable	Tentative	62.0	32.5	3.0	2.5	5.5
Semantic	Variable	Variable	62.0	31.7	3.1	3.2	6.3
Semantic	Semantic	Tentative	60.3	32.6	3.7	3.4	7.1
Semantic	Semantic	Variable	60.8	32.8	2.1	4.3	6.4
Semantic	Semantic	Semantic	61.0	39.0	0.0	0.0	0.0

the purposes of this investigation, but the logic could be much more complicated in a real world system (for example, a provider could always offer semantic atomicity to regular clients, or only ever offer tentative holds to unfamiliar clients with large orders).

It is assumed that each provider offers the same number of resources. This is tested firstly with each provider offering 1000 resources, and then with each offering 3500. Thus, in the first test, there are far fewer resources than are required by the set of clients, meaning that the reservations for these providers are most likely to fail. When each provider has 3500 resources, there are sufficient resources for each client and transactions only fail if the other actions fail, or semantic atomicity is required but not offered.

5 RESULTS

The results for the test where providers only have 1000 resources are shown in Table 1. In each case, at least 99.9% of the resources offered by each provider were used, so the only important difference between tests are the protocols used by the providers and the number of clients that succeeded, failed without penalty, and failed with penalty.

The protocol offered by each provider is specified in its "Protocol" column of the table. "Tentative" means that the provider offered the *tentative scheme*, which only ever gives out tentative holds. "Semantic" means that the provider offered the *semantic scheme*, always offering semantic atomicity if the requested resources are available. Finally, "Variable" means that the provider offered the *variable scheme*, where it switches from offering semantic atomicity to offering tentative holds once its change-over threshold has been reached.

The other columns in the table show the success or failure rate of the clients. The "Success" column gives the percentage of clients that successfully booked the required resources from one of the three providers and also had their other activities succeed. The "No penalty" column specifies how many clients had both actions fail without penalty. "Penalty on other activity" gives the number of clients that had their other activities succeed but were unable to book the resources required from any of the three providers. "Penalty on this service" gives the number of clients that successfully booked the resources from one of the providers, but then had their other activities fail. "Any penalty" is the sum of "Penalty on other activity" and "Penalty on this service", and indicates how many clients had a failure resulting in a partially completed transaction.

When all providers offered the *tentative scheme*, 61.2% of transactions completed successfully, and 10.1% failed with penalty. When a single provider instead offered the *variable scheme*, the number of failures with penalty was slightly lower (8.8%). Compare this to the case where a single provider offered the *semantic scheme* while the others offered the *tentative scheme*. In this case, only 6.5% of transactions failed with penalty. This shows that the *variable scheme* provides better service for clients than exclusive use of the *tentative scheme*, as the rate of client failures with penalty is much lower (though not as low as when the provider offers the *semantic scheme*).

Further, replacing more providers offering the *tentative scheme* with providers that offer the *variable scheme* results in even lower rates of failure with penalty. When all providers offered the *vari-*

Table 2: Results when providers have sufficient resources.

Provider 1		Provider 2		Provider 3		Clients		
Protocol	Utility (%)	Protocol	Utility (%)	Protocol	Utility (%)	Success (%)	Failure with no penalty (%)	Failure with penalty (%)
Tentative	37.37	Tentative	34.49	Tentative	32.17	72.6	18.8	8.6
Variable	68.89	Tentative	19.71	Tentative	20.06	76.6	18.0	5.4
Variable	53.57	Variable	54.46	Tentative	7.51	80.0	18.8	1.2
Variable	43.31	Variable	37.60	Variable	35.40	80.7	19.3	0.0
Semantic	99.40	Tentative	06.91	Tentative	8.11	80.0	19.2	0.8
Semantic	63.66	Variable	49.51	Tentative	4.11	80.7	18.8	0.5
Semantic	43.31	Variable	37.60	Variable	35.40	80.7	19.3	0.0
Semantic	59.09	Semantic	54.09	Tentative	4.11	80.7	18.8	0.5
Semantic	43.31	Semantic	37.60	Variable	35.40	80.7	19.3	0.0
Semantic	43.31	Semantic	37.60	Semantic	35.40	80.7	19.3	0.0

able scheme, only 6.7% of transactions failed with a penalty for the client. While this is not as good as the 0% rate when only the *semantic scheme* was offered, it is significantly better than the original 10.1%.

The success rate of transactions using techniques other than the *tentative scheme* are also fairly high. The decreased risk obviously means that fewer transactions can succeed, but this is almost completely compensated for by the clients that refuse to use any transactional guarantee other than semantic atomicity; when only the *tentative scheme* is offered, they are guaranteed to fail (without penalty), but when even just one provider offers the *semantic scheme* or the *variable scheme* then these transactions have a chance to succeed.

Table 2 shows the results of starting each provider with 3500 available resources. Again, clients can be seen to have a lower chance of failure with penalty when transactional guarantees stronger than tentative holds are provided. In fact, as there are sufficient resources, the only failures with penalty that occur are when the client risks making a booking on a provider that only offers a tentative hold, and then the other actions in the client's transaction fail.

More interesting are the results for provider utility, namely the percentage of original resources that the clients have used once all transactions have completed. Since it is assumed that most clients prefer using a provider who offers semantic atomicity, when each provider has enough resources, those using the *variable scheme* or the *semantic scheme* have a higher utility. In fact, in many cases, the provider offering the *variable scheme* behaves exactly like the provider offering the *semantic scheme*, as the number of resources it has remaining never gets below the amount where the switch to tentative hold occurs.

The most obvious exception to this is where two of the providers offer the *tentative scheme*, and the other offers either the *variable scheme* or the *semantic scheme*. When the provider offers the *variable scheme*, it has a utility of 68.89%. When the *semantic scheme* is used instead, however, the provider has a utility of 99.40%. This is because the provider offering the *variable scheme* offers semantic atomicity until its utility would be over 50%, and then only offers tentative holds. Thus, once the provider changes to tentative holds, clients no longer prefer that provider over any of the others. When the *semantic scheme* is offered, however, more clients prefer using that provider as long as it has resources available.

Finally, table 3 shows the results when each provider offers 3500 resources, and half the clients attempt to order 50 of them. Once again, the number of transactions that fail with penalty decreases as more providers offer stronger transactional guarantees. However, since many clients attempt to book large numbers of resources, when only the *semantic scheme* is offered the utility of each provider is significantly reduced. This is because clients lock the resources, making them unavailable for others, so more transactions fail immediately. Interestingly, when just one provider offers the *variable scheme* or the *tentative scheme* rather than the *semantic scheme*, the utility of all providers is increased. Since clients get the tentative holds rather than immediately failing, the transaction stays active for longer, giving it a greater chance to succeed. If ever the tentative hold is cancelled, the client, rather than simply failing, searches for another provider to offer the required resources. If this occurs after a transaction that had an exclu-

Table 3: Results when half of the clients request large amounts of resources.

Provider 1		Provider 2		Provider 3		Clients		
Protocol	Utility (%)	Protocol	Utility (%)	Protocol	Utility (%)	Success (%)	Failure with no penalty (%)	Failure with penalty (%)
Tentative	98.51	Tentative	99.91	Tentative	100.00	46.5	32.6	20.9
Variable	99.94	Tentative	99.54	Tentative	98.74	45.0	36.3	18.7
Variable	99.11	Variable	99.89	Tentative	99.74	43.4	43.0	13.6
Variable	98.06	Variable	99.49	Variable	99.57	43.0	48.0	9.0
Semantic	99.94	Tentative	98.89	Tentative	99.54	44.9	41.6	13.5
Semantic	99.94	Variable	100.00	Tentative	97.00	44.8	45.9	9.3
Semantic	99.94	Variable	99.71	Variable	99.34	42.2	53.4	4.4
Semantic	99.69	Semantic	99.97	Tentative	99.80	41.3	51.5	7.2
Semantic	97.60	Semantic	96.31	Variable	96.14	39.7	57.6	2.7
Semantic	83.06	Semantic	81.34	Semantic	83.51	34.8	65.2	0.0

sive hold on items from a different provider has failed, those resources are then available to the client that lost the hold.

6 CONCLUSIONS AND FUTURE WORK

Transactions in the Web Services environment are necessarily quite different to traditional ACID transactions. However, the support provided by current standards is not always sufficient. While existing standards do allow some reduction to the ACID properties, these reductions are static, and each provider in the transaction must necessarily support the same reductions.

We have demonstrated that allowing service providers to dynamically decide on the level of transaction support to offer for a particular request can be beneficial to both service providers and clients. Using a simple example where a provider changes from offering semantic atomicity to tentative holds based purely on the number of resources it has available, it has been seen that fewer client transactions fail with a penalty. Further, assuming that most users would prefer services with better transactional guarantees, the utility of the provider increases.

It has also been shown that, in certain circumstances, using the variable scheme can increase the utility of providers over that supplied by using only semantic atomicity. Thus, the alternate scheme has benefits over both offering just semantic atomicity and just tentative holds.

While the example in this paper only considers two different transactional approaches (semantic atomicity and tentative hold), many others are available. Thus, future work will look at combining different transactional guarantees into a single system in which providers can dynamically decide on the level of transaction support they wish to provide. This study will also investigate different logic to help providers make that decision.

The authors have not discussed the actual protocol to allow clients and providers to negotiate on the level of transaction support to be provided for a particular service request. When performing a transaction comprised of actions from multiple providers, the level of risk a client is willing to accept on one service call may depend on the guarantees given by the providers of the other actions in the transaction. Thus, not only will the level of support a provider is willing to guarantee change dynamically, but so too will the level that a client is willing to accept. As service providers typically want their services to be used to give the highest profit possible, and offering certain transactional guarantees can be expensive, research into determining how a provider can discover the level of support required to attract a particular client will be as important as having the client determine what level is being offered.

REFERENCES

Alrifai, M., Dolog, P., and Nejdl, W. (2006). Transactions concurrency control in Web Service environment. In *The Fourth IEEE European Conference on Web Services (ECOWS'06)*, pages 109–118.

Bunting, D., Chapman, M., Hurley, O., Little, M., Mischkinsky, J., Newcomer, E., Webber, J., and Swenson, K. (2003). Web Services Composite Application

Framework (WS-CAF) ver1.0. Technical report, Arjuna Technologies Ltd.

Cabrera, L. F., Copeland, G., Feingold, M., Freund, R. W., Freund, T., Johnson, J., Joyce, S., Kaler, C., Klein, J., Langworthy, D., Little, M., Nadalin, A., Newcomer, E., Orchard, D., Robinson, I., Shewchuk, J., and Storey, T. (2005). Web Services Coordination (WS-Coordination). Technical report, Arjuna Technologies Ltd., BEA Systems Inc, Hitachi Ltd., IBM Corporation, IONA Technologies, Microsoft Corporation.

Ceponkus, A., Dalal, S., Fletcher, T., Furniss, P., Green, A., and Pope, B. (2002). Business Transaction Protocol 1.0. Technical report, OASIS.

Choi, S., Jang, H., Kim, J., Kim, S. M., Song, J., and Lee, Y. (2005). Maintaining consistency under isolation relaxation of Web Services transactions. In *The Sixth International Conference on Web Information Systems Engineering (WISE'05)*, pages 245–257, New York, NY, USA.

Cox, W., Cabrera, L. F., Copeland, G., Freund, T., Klein, J., Storey, T., and Thatte, S. (2004). Web Services Transaction (WS-Transaction). Technical report, BEA Systems Inc, International Business Machines Corporation, Microsoft Corporation.

Fauvet, M.-C., Duarte, H., Duman, M., and Benatallah, B. (2005). Handling transactional properties in web service composition. In *The Sixth International Conference on Web Information Systems Engineering (WISE'05)*, pages 273–289.

Garcia-Molina, H. (1983). Using semantic knowledge for transaction processing in a distributed database. *ACM Trans. Database Syst.*, 8(2):186–213.

Gray, J. and Reuter, A. (1993). *Transaction processing : concepts and techniques*. Morgan Kaufmann Publishers, San Mateo, Calif.

Limthanmaphon, B. and Zhang, Y. (2004). Web Service composition transaction management. In *The Fifteenth Australasian database conference (ADC '04)*, pages 171–179. Australian Computer Society, Inc.

Lyon, J., Evans, K., and Klein, J. (1998). Transaction Internet Protocol version 3.0. Technical report, Microsoft Corporation and Tandem Computers.

Mikalsen, T., Tai, S., and Rouvellou, I. (2002). Transactional attitudes: Reliable composition of autonomous web services. In *Workshop on Dependable Middleware-based Systems (WDMS'02) at the Dependable Systems and Network Conference (DSN'02)*.

Roberts, J. and Srinivasan, K. (2001). Tentative Hold Protocol part 1: White paper. Note, World Wide Web Consortium.

Schäfer, M., Dolog, P., and Nejdl, W. (2007). Engineering compensations in web service environment. In *International Conference on Web Engineering*, Como, Italy. Springer Berlin/Heidelberg.

Zhang, A., Nodine, M., Bhargava, B., and Bukhres, O. (1994). Ensuring relaxed atomicity for flexible transactions in multidatabase systems. In *The ACM SIGMOD International Conference on Management of Data (SIGMOD'94)*, pages 67–78. ACM Press.