SIMULATING WEB SERVICES TRANSACTIONS

David Paul, Frans Henskens and Michael Hannaford
School of Electrical Engineering and Computer Science, University of Newcastle, Callaghan 2308, NSW, Australia

Keywords: Web service, Transaction, Simulation, Protocol, Performance, Concurrency.

Abstract: Transactions in a service-oriented environment are very different to traditional transactions. The typical ACID properties are not always appropriate when processing transactions where multiple service providers wish to maintain autonomy and process all requests in a timely manner. Thus, reductions to the ACID properties, such as semantic atomicity and tentative holds, are often used to provide transactions for Web Services. This paper describes a simulator for modelling various Web Services transaction strategies. The simulator is deterministic, allowing the specification of network conditions, provider resources, and client workflows to be kept constant while altering the level of transaction support each provider offers. By modelling transaction flow rather than service flow, this allows true comparison of transaction techniques in various scenarios. The simulator is demonstrated using a validation experiment, and future use outlines how the simulator is being used to test a system where providers offer a dynamic level of transaction support to clients.

1 INTRODUCTION

Service-Oriented Architectures (SOAs) are based on service providers offering services to clients without clients needing to understand the intricate details of how a particular service is performed. Typically, the client simply sends a request asking for a service to be performed, and the provider either replies with a message that the request was unsuccessful, or performs the service and informs the client that the request has completed successfully. Thus, the client only sees whether the service has been performed, with no further idea of how the provider completed the required tasks. This is a powerful model that overcomes the tight-coupling inherent in other distributed computing paradigms (Henning, 2006).

One of the most common implementations of an SOA is that of Web Services (Tiezzi, 2009). Using open standards (Christensen et al., 2001; Clement et al., 2004; Gudgin et al., 2007), Web Services allow practically any party on the Internet to access and utilise service-oriented interactions with Web Services providers, regardless of any hardware, software, or other environmental differences between the communicating hosts. Further, these standards can be used as the basis of new standards to add support for features not included in the original standards. For example, Web Services Security (WS-Security) (Lawrence and Kaler, 2006) provides message integrity and confidentiality to Web Services.

While SOAs allow easy abstraction when a client only wishes to perform one service, it is often the case that a client will require a workflow that utilises multiple services from different providers to achieve the results that the client desires (Barker et al., 2009). The various providers used by such a client may have no other connection than their services being needed for that client's workflow. Thus, the client requires a method to combine these disparate services in such a way as to ensure correct processing of the workflow. In particular, some assurances of the outcome of the workflow when some providers are unable or unwilling to provide the services required of the client are needed. Transactions provide a possible way to offer such assurances (Papazoglou, 2003).

In traditional systems, transactions comply with the ACID properties (Atomicity, Consistency, Isolation, and Durability) (Gray and Reuter, 1993). However, complete support for these properties is not always possible or desirable in the Web Services environment (Little, 2003). Instead, slightly weaker transactional properties are typically preferred in Web Services transactions to ensure both provider autonomy and a reasonable quality of service (Buckpesch and Maur, 2006).

While various alternatives for Web Services transactions have been defined (Lyon et al., 1998; Younas et al., 2000; Ben Lakhal et al., 2001; Roberts and

Paul D., Henskens F. and Hannaford M..
SIMULATING WEB SERVICES TRANSACTIONS.
DOI: 10.5220/0003472206150623
In *Proceedings of the 7th International Conference on Web Information Systems and Technologies* (WSPA-2011), pages 615-623
ISBN: 978-989-8425-51-5

Srinivasan, 2001; Ceponkus et al., 2002; Milanovic et al., 2003; Bunting et al., 2003; Lin and Liu, 2005; Wang et al., 2007), there has been very little testing to compare the possible types of transactions to see how they perform and to determine strategies to optimise transaction use in the Web Services environment. This paper describes a simulator to allow various different levels of transaction support to be directly compared. The simulator models the flow of transactions rather than the flow of services. A validation experiment is also presented to show how the simulator performs.

The remainder of the paper is organised as follows. Section 2 describes in more detail the differences between traditional transactions and those in the Web Services environment. Section 3 then describes the Web Services transactions simulator. The validation experiment is then described in Section 4, and results are given in Section 5. Finally, Section 6 concludes the paper and indicates some future research directions.

2 TRANSACTIONS IN THE WEB SERVICES ENVIRONMENT

The traditional ACID transaction properties usually require either pessimistic or optimistic concurrency control to ensure that concurrent transactions do not interfere with each other. When a pessimistic scheme is used, resources can be locked for long periods of time. When optimistic schemes are used, many transactions may be required to redo work that they have already completed if another transaction interferes with them. Neither of these are greatly appropriate in the Web Services environment (Little and Freund, 2003), where transactions can take days to complete. Further, traditional systems usually have a centralised controller that is in charge of all entities involved with the transaction. Such a central controller does not exist in service-oriented environments, where each service provider need not even know that other providers exist, let alone have any control over them.

Very few providers would be willing to give up the autonomy necessary to support traditional transaction schemes. Each service provider is an individual entity with complete control over its actions. While clever techniques do allow full support for the ACID properties in the Web Services environment (Choi et al., 2005; Alrifai et al., 2006), these still require service providers to lock resources for, potentially, long periods of time. Further, the length of time required for any such lock is highly variable because it depends on the workflow of the client requesting the resource. Thus, the ACID properties are often inappropriate for Web Services transactions.

2.1 Reductions to ACID Properties

Various reductions to the strength of some of the ACID properties have been suggested. These include semantic atomicity (Garcia-Molina, 1983), tentative holds (Roberts and Srinivasan, 2001), and resilience (Younas et al., 2006). A brief explanation of these reduced properties is given below.

2.1.1 Semantic Atomicity

Semantic atomicity replaces the traditional atomicity property. Rather than requiring transactions to always appear as either completed or not started, semantic atomicity only requires that a transaction eventually resolves into one of those two states. Semantic atomicity is based on the concept of compensating actions (Levy et al., 1991), which logically undo previously completed actions. Thus, when a provider is requested to perform an action, the action is processed immediately. If either the action fails, or the action succeeds and is never cancelled, then the behaviour is identical to the behaviour had atomicity been enforced. However, if the action succeeds and the transaction is later cancelled, then the compensating action for the performed action is carried out. This compensating action performs any steps required to make it appear as if the original request had never occurred. In between the time when the original action is performed and the compensating action completes, other transactions may see the action as having been performed when logically they should not. Thus, semantic atomicity necessarily removes isolation.

2.1.2 Tentative Holds

Another commonly used reduction for the traditional ACID properties is the support for tentative holds (Fauvet et al., 2005). Tentative holds can be thought of as advice that the requested resources are currently available, without any guarantee that the resources will be available in the future. A common example are items in an online shopping cart. Multiple users can have the same item in their cart, but when one checks out and purchases the item then all other holds on the item are cancelled, and the users are informed of this fact. When a client has tentative holds on all the resources required for its workflow it is certain that, at one point in time, the workflow could have succeeded. In such a case, it is possible that some of the resources will become unavailable before the client is able to convert the tentative holds to confirm-

ed holds, but the client has increased confidence that the transaction can succeed.

2.1.3 Resilience

Resilience allows the transaction to try alternatives rather than failing as soon as a service call is unsuccessful. Resilience may repeatably request an operation until it succeeds, or may search for alternative ways to achieve a goal. For example, when booking a holiday, if no acceptable flight is available with a particular airline, a resilient system may book a flight with another airline instead.

2.2 Describing Reductions

It has been suggested that the ACID properties should be replaced with the SACReD properties (Younas et al., 2006) in the Web Services environment. Rather than being an individual reduction in the strength of ACID properties, the SACReD properties take advantage of some of the unique features of SOAs to offer an alternative. The C and D stand for consistency and durability, as in the traditional ACID properties. SA stands for semantic atomicity, and Re adds the concept of resilience. However, the SACReD properties cannot describe all of the possible reductions in transactional strength. For example, the SACReD properties do not include tentative holds.

The reductions described in Section 2.1 are the most commonly used reductions to the ACID properties used for Web Services transactions, but there are others. Examining the possible reductions, most can be described by using a combination of five basic operations that can be requested by a client:

Enquire. Allows the client to query whether a request would currently be successful, without any guarantee that a later request will succeed.

Prepare. Allows the client to query whether a request would currently be successful and, if so, guarantees that any such request sent by the client within a timeout period will succeed. On receipt of a successful reply, the client has the option to cancel the request, which relieves the provider of its responsibility to guarantee the resources to the client.

Commit. Performs the client's request. This is the only required operation for a service.

Compensate. Performs actions to undo a previously committed request, provided that the call to compensate is received within a timeout period.

Callback. Allows a provider to notify a client that has previously received a response from the provider in the case that the provider's situation has changed, changing the provider's response.

Using these five operations, a traditional ACID transaction can be described as a Prepare/Commit pattern with an infinite timeout for the Prepare stage. Similarly, semantic atomicity is provided by having the provider offer a Commit/Compensate pattern, and support for tentative holds is achieved through an Enquire/Callback/Commit pattern.

The concept of resilience cannot be described using these operations. However, replacing concrete services with abstract services (Schäfer et al., 2007) can allow resilience in a way that is transparent to both clients and service providers. Clients use the abstract services rather than the services offered by the concrete providers, and the abstract service acts as a broker between all the providers offering alternative services. These abstract services can thus be used in transactions described by the operations defined above.

2.3 Combining Different Levels of Transactional Support

While the different transaction reductions do greatly help make transactions more useful in the Web Services environment, the use of a single reduction for every action in a transaction is not always adequate (Greenfield et al., 2003). Combining different reductions together is possible (Mikalsen et al., 2002; Paul et al., 2010), and results in systems in which both clients and providers are better able to have their needs met. However, strategies on how best to support and combine different transactional properties have not been thoroughly investigated. The simulator described in this paper enables such strategies to be tested much more easily than other current techniques.

3 SIMULATING WEB SERVICES TRANSACTIONS

Web Services transactions are complex (Paul et al., 2008). There are many different interacting entities, all working autonomously but cooperatively, and therefore verification and testing of transaction schemes is also very complex. This makes it difficult for theoretical analysis alone to yield practical results. Simulation allows the practical characteristics of the various transaction schemes to be compared with much more control and at a much lower cost than is possible with real-world testing.

Analytical results for Web Services transactions are important; in particular, it is vital to verify that the various reductions to the ACID properties work correctly. Theoretical models have been developed to allow such verification (Fantechi et al., 2009; Mayer et al., 2008; Lapadula et al., 2008), but these models do not easily allow direct comparison between different transaction techniques. The models can indicate whether techniques are valid, but cannot always indicate the application for which (and environment in which) a technique is most beneficial. We believe an alternate method is required to allow evaluation of the real-world performance characteristics of transaction schemes.

To test the performance of Web Services transaction techniques, one option is to set up real systems and conduct experiments utilising the different schemes. A major problem with this approach is that it requires large networks to be set up to perform the tests. Setting up such an environment is costly, and any results may be dependant on uncontrollable variables that are not related to those being tested. While it may be possible to set up a system in a local environment, Web Services are often used to cross boundaries between organisations, and across large geographical areas. The results obtained through a local experiment are only applicable to the environment in place at the time of the experiment. Similarly, if a multi-organisational network were deployed for transactional testing, any results would be dependant on conditions of the network. For example, consider a comparison of two different transaction techniques where the first is tested at a time when there is a significantly higher amount of unrelated network traffic than when the second is tested. Any comparison may unfairly discriminate against the first technique because of the higher network load.

Instead, simulation can be used as a low-cost addition to analysis, and will yield results that allow intricate comparison of particular transaction strategies. In a simulation, some or all of a system is abstracted so that only the features important to the current investigation are tested (Kelton et al., 2004). To simulate Web Services transactions, details such as network topology, the timing of various events, and even the actual services being performed can be abstracted to isolate the parameters of interest and allow more efficient study.

Most available Web Services simulation environments replace a Web Service with a simple, usually local, program that answers responses in a way appropriate to the service being simulated (Winston, 1999; Kilgore, 2003). This allows the flow of services to be tested. However, when examining Web Services transactions, further abstraction can occur. Only the transaction interaction patterns need be simulated; the messages need not be exactly formatted as for a particular service. Section 3.1 describes a simulator that models transaction flow rather than service flow.

3.1 The Web Services Transaction Simulator

The Web Services simulator is composed of two distinct parts: the generator and the simulator. The generator allows the creation of descriptions of scenarios that are used as input to the simulator. The simulator uses the input description and other parameters (described in Section 3.1.2) to determine the outcome of the scenario. Thus, by keeping the scenario description constant and varying the simulator's other parameters, the effect of the changed parameters can be studied.

This paper concentrates on the simulator only. Section 3.1.1 describes the details of the scenario descriptions that the simulator takes as input. Section 3.1.2 then describes the other parameters to the simulator. Details of the operation of the simulator are given in Section 3.1.3, and Section 3.1.4 describes the simulator's output.

3.1.1 Describing Scenarios

A scenario description contains information about the concrete participants in the scenario. The service providers are specified; each provider allows clients to book resources for a price. The number of resources offered by a particular provider can be limited or infinite, and booking a resource can either be free of charge, or at a cost. The likelihood that a provider will agree to perform the service when sufficient resources are available is also part of the description.

To allow resilience, abstract providers can then be specified in a service description. Abstract providers contain a list of concrete providers. When an abstract provider receives a service request from a client, it forwards that request to one of the listed concrete providers. If the request fails then, rather than notifying the client of the failure, the abstract provider requests the service from another of the concrete providers in its list. The client only receives a notification of failure if all of the concrete providers that the abstract provider contacts refuse to perform the service.

After provider information, a scenario description contains details of a set of clients and the workflows they wish to complete. Each workflow consists of a sequence of activities to be performed, and specifies

whether successful completion of each activity is required, or if the workflow is successful when at least one of the included activities succeeds. When all activities are required, the workflow fails whenever one or more included activity fails. Otherwise, the workflow only fails when all included activities complete unsuccessfully. An activity can either be a workflow or a service call. A service call consists of the name of a provider, the name of the service to be requested from that provider, and the number of resources to request from the service.

Finally, a scenario description contains timing information. When sending messages between clients and providers, timing is very important. Messages are not received immediately after they are sent, and the length of time required for a message to travel between two parties is dependant on many factors, such as network bandwidth and congestion, and the load of the systems involved in the communication. Similarly, once a message is received, it takes time for the message to be processed. The scenario description includes a set of time modellers that specify how long messages take from when they are first sent to when they begin being processed, as well as how long the processing of the message takes. There is a time modeller for each pair of communicating participants, allowing simulation of the large range and rapid changes possible in both network conditions and the local conditions of the participants involved.

3.1.2 Simulator Parameters

As well as the scenario description, the simulator also has parameters to control the level of transaction support offered by the service providers and the risk-taking behaviour of the clients. Different transaction strategies and techniques can thus be tested on an identical scenario by varying the simulator's other parameters. The simulator is deterministic, so any change in results can be attributed to the change in parameter values.

The transaction support offered by concrete providers is specified based on the five operations defined in Section 3.1.1. Combined with the abstract services in the scenario description, this allows providers to utilise all of the reductions to the standard ACID properties. The level of transaction support offered by an abstract service depends on the levels of support offered by the concrete providers that it contacts. While the simulator does allow abstract services to offer varying levels of transaction support based on the client's requirements and the levels of support offered by the concrete providers, details of this are beyond the scope of this paper.

Another important set of parameters for the simulator is the risk-taking behaviour of the clients. A client's risk-taking behaviour determines how likely it is that the client will perform an action that has the possibility of leaving a client in a worse state than when it started. For example, a client with high risk-taking behaviour would simply ask all parallel operations to commit simultaneously, without regard to the fact that some requests may succeed and others may fail. Alternatively, when minimising risk, a client may first only call services that offer a Prepare or Compensate stage so that, if any action fails, the semantic atomicity of the transaction can be maintained.

3.1.3 Simulator Operation

Web Services are based on a messaging model by which a message is sent from one party to another and is then processed by the receiver. On receipt of the message, processing may alter the receiver's state, and may also send messages either in reply to the original sender or on to a third party. The Web Services transaction simulator follows this messaging model. When a client or provider wishes to send a message, it notifies the simulator. The simulator uses the time modeller associated with the sender and receiver of the message to determine when the message should be delivered. All messages are added to a priority queue and the simulator ensures that each message arrives at its destination at the correct time.

When a provider has only finite resources to offer, the simulator tracks the state of the provider's interaction with each client that sends it a request. This allows correct transactional behaviour. For example, when an ACID Prepare/Commit pattern is offered, the provider must not offer any resources that it has guaranteed to be available to one client to any other client until the first client (that requested the resources) has finished its transaction. If a tentative hold Enquire/Callback/Commit pattern is used instead, the provider can have multiple clients with tentative holds on some of its resources, but if one of those clients request to commit then the provider must send a notification to the other clients that their hold is no longer valid.

Similarly, under the simulator, clients monitor their interactions with providers. Each client tracks the progress of the various activities in its workflow. The risk-taking behaviour of the client is continually examined to determine how the client should proceed. This allows three possibilities, depending on whether the risk of continuing is judged to be acceptable, unacceptable, or uncertain. If the risk is acceptable, the client sends messages to begin its next action. If the risk is unacceptable then the client cancels as much of the processed workflow as is possible and the transac-

tion fails. If the client is uncertain of the risk of continuing then it waits until its situation changes before taking any further action.

3.1.4 Simulator Output

Simulation continues until all clients and providers have finished processing. As the simulation is executing, a log of all messages is created. This includes the name of the sender and receiver of the message, the time the message was sent, the time it is received, and the content of the message. Combined with the simulator parameters, the message log allows various details to be extracted. This includes information such as the length of a client's transaction, whether the client's workflow completed successfully, and the cost to the client. From the provider's point of view, the number of resources utilised and the amount clients paid to the provider can be measured. These results provide the necessary details to allow comparison of the various transaction strategies.

4 VALIDATION EXPERIMENT

This section describes a validation experiment to demonstrate the simulator's use. The experiment is designed to show the shortcomings of the ACID properties in SOAs (such as thoat provided by Web Services).

4.1 Simulation Parameters

This scenario tests a single provider offering 1000 resources with four possible levels of transaction support. The four levels of transaction support tested are a traditional ACID Prepare/Commit scheme, a traditional ACID Prepare/Commit scheme with a timeout of 500 (so that any call to begin automatically cancels if not told to commit within 500 time units of when the first call was received), a tentative hold Enquire/Callback/Commit scheme, and a semantic atomicity Commit/Compensate scheme. Clients in this scenario wish to book resources from the provider, and complete some other actions. For the purpose of this scenario, there are 1000 clients who each wish to book between 1 and 10 (randomly chosen) resources from the provider. The other actions are modelled as another service call that takes between 1 and 50 time units to complete, and has a 20% likelihood of failure.

4.2 Assumptions

The different transaction types can only be properly tested when there is resource contention. When resources are plentiful, each client can access the resources they require without affecting other concurrent transactions. When resources are limited, but timing of transactions is spread so that none are executing concurrently, the first transactions will successfully utilise all of the available resources, and later transactions will fail simply because the resources they require are unavailable. Since the 1000 clients are requesting well over the 1000 resources available from the provider, it is sufficient to ensure that the transactions are executing concurrently to create true resource contention. To this end, each client's transaction is set to start between time 0 and time 100; as each transaction can take over 50 time units, this ensures that resource contention occurs.

As mentioned in Section 3, events do not occur immediately when a message is sent; each message takes measurable time to travel from the sender to the recipient. For the purposes of this experiment, it is assumed that each message takes between 1 and 5 time units from when it is sent to when its processing commences. The processing of a message is also not instantaneous; the provider in this scenario takes between 1 and 10 time units to process a request.

5 RESULTS

The results are presented in Table 1. The first column indicates the level of transaction support being tested. The "provider utility" column gives the percentage of resources originally offered by the provider that were booked once all transactions had completed. The "client successful" column indicates how many transactions completed all operations successfully. This is in contrast to "client unsuccessful without penalty", which shows how many transactions failed without completing any of the required operations, and "client unsuccessful with penalty", which shows the number of transactions that end partially completed, with either the resource booking completing successfully but the rest of the transaction failing, or the resource booking failing and rest of the transaction completing successfully. The final column indicates the average duration of the client transactions for each of the different levels of transaction support.

In each case, except where semantic atomicity is offered, all of the provider's resources are utilised. As the number of client transactions is so large and the number of resources offered by the provider so

Table 1: Results for Validation Experiment.

Transaction support	Provider utility (%)	Client successful (%)	Client unsuccessful without penalty (%)	Client unsuccessful with penalty (%)	Average duration of a transaction
ACID	100	23.0	77.0	0.0	1316
ACID with time out	100	24.5	72.8	2.7	1245
Tentative hold	100	15.6	57.5	26.9	228
Semantic atomicity	86.2	16.7	83.3	0.0	134

small, this is hardly surprising; many more resources would be required to satisfy all clients' requests, so there are many clients all wishing to access the same resources. When the ACID schemes are used, the resources quickly become locked by the first clients requesting them, and all other clients must wait until a final decision (or time out) occurs before discovering whether they have access to the resources they require. Thus, if ever a client decides against using the resources it had locked, there are numerous other clients waiting to place a lock on the resources freed by that client. Similarly, when the tentative hold scheme is used, numerous clients have simultaneous holds on the resources. Since those holds are only invalidated when resources are actually booked, transactions only fail if the other actions in the transaction fail, or the requested resources are truly not available. In contrast, when semantic atomicity is offered, it is possible for a transaction to fail because another client's transaction has already booked those resources. If the other transaction later compensates that booking, however, then the first transaction may have been able to succeed, but, since the resources were not available when that transaction needed them, the transaction has already failed.

Looking at the results of the clients' transactions, more transactions succeed when an ACID scheme is used. This is because clients wait until they are guaranteed their request for resources will be successful before attempting the remainder of the transaction. This essentially reduces the number of concurrently executing transactions to only those that have access to the resources they require; all other executing transactions wait to be given access to the resources before they continue with the rest of their transaction. When a timeout is included, it is possible that a client may have the required resources locked and then attempt to complete the rest of the transaction, but have the timeout occur before they can complete their booking. This results in a failure with penalty, as the other actions may have completed and not be cancellable, though the longer the timeout period the less likely this would be. On the other hand, when tentative hold is used, many clients are granted holds on the same resources and then attempt to complete the rest of their transaction. Thus it is much more likely that transactions fail with a penalty, as they complete the rest of their transaction, but their tentative hold expires before they can finalise their booking of the resources. When semantic atomicity is used, as stated in the previous paragraph, many transactions fail at an earlier time when they could have succeeded later, which reduces the number that complete successfully. However, as when an ACID scheme with no timeout is used, transactions only ever fail without penalty.

Thus, traditional ACID techniques are much better at ensuring successful transaction completion utilising as many resources as possible. However, considering the average duration of a transaction, it can be seen why they are not preferred for transactions in a service-oriented environment. Forcing clients to wait until a guarantee that the requested resources will be available means that clients spend a lot of their transaction time waiting. When a traditional scheme is used, the average duration of a transaction is over 1200 time units, compared to only 228 when tentative hold is used, or 134 when semantic atomicity is offered. In many cases, clients would not be willing to wait that long, and would thus not use the provider offering the traditional scheme. These results confirm the accepted knowledge that strict adherence to the ACID properties is typically not suitable for SOA transactions (Dalal et al., 2003).

6 CONCLUSIONS

The environment offered by SOAs is very different to that typically utilised by traditional transactions. The high autonomy of all service providers and the potential for transactions to run for long periods of time mean that the typical ACID properties are not always appropriate. Instead, different techniques, such as tentative holds and semantic atomicity, which redu-

ce the strength of some of the ACID properties, are used to avoid some of the negative effects introduced by strong transactional support. Thus, there is a balancing act between the level of transaction support offered and the autonomy of the service providers involved in a transaction.

The various reductions used for Web Services transactions have different strengths and weaknesses. However, it is difficult to compare the performance of the various possible reductions, as real-world testing requires large infrastructure and makes truly repeatable tests impossible. While analytical approaches are important to show the viability and correctness of the various transaction schemes, it is difficult to compare all schemes using these methods, as they often lack real-world performance measures. Simulation can build on the theory to give an indication of practical results without the difficulties associated with real-world testing. Various scenarios can repeatably be processed by the simulator to compare changes to the transaction scheme being used, and the practical viability of the different schemes can be evaluated.

This paper describes a simulator that models the flow of transactions to allow testing of different Web Services transaction techniques. By abstracting over details such as network setup, timing of messages, and the actual operations being performed, the simulator is able to utilise different transaction standards for various scenarios to determine how the transaction techniques perform in the given circumstances. The simulator is deterministic, allowing true comparison of the transaction techniques when performing identical operations. This allows much easier evaluation of the various transaction options, which can lead to a more informed choice of transactional support for real-world services.

To demonstrate the simulator's functionality, a simple scenario was modelled to show how ACID transactions compare to transactions with either semantic atomicity or tentative hold support. This showed that ACID transactions can allow more successful completions of transactions, but do this at the cost of time. The average duration of ACID transactions in this simulation was over five times longer than the average time when the weaker transaction guarantees were used. This shows that, in environments such as that of Web Services, where long transaction times should be avoided, ACID transactions are often not the best choice.

In the future, the simulator will be used to compare different transaction strategies for service providers. This includes more extensive research into the use of dynamic transaction support, where the level of transaction support offered by a service provider varies based on the provider's current circumstances (Paul et al., 2010). This study will allow both clients and service providers to negotiate the level of transaction support being offered to ensure that all parties involved are satisfied with the results.

REFERENCES

Alrifai, M., Dolog, P., and Nejdl, W. (2006). Transactions concurrency control in Web Service environment. In *The 4th IEEE European Conference on Web Services (ECOWS'06)*, pages 109–118.

Barker, A., Walton, C., and Robertson, D. (2009). Choreographing Web Services. *IEEE Transactions on Services Computing*, 2(2):152–166.

Ben Lakhal, N., Kobayashi, T., and Yokota, H. (2001). WS-Sagas: Transaction Model for Reliable Web-Services-Composition Specification and Execution. *DBSJ Letters*, 2(2):17–20.

Buckpesch, S. and Maur, M. (2006). Transactions in Web Services. Technical report, Darmstadt University of Technology.

Bunting, D., Chapman, M., Hurley, O., Little, M., Mischkinsky, J., Newcomer, E., Webber, J., and Swenson, K. (2003). Web Services Transaction Management (WS-TXM) ver1.0. Technical report, Arjuna Technologies Ltd.

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

Choi, S., Kim, J., Jang, H., Kim, S. M., Song, J., Kim, H., and Lee, Y. (2005). A framework for handling dependencies among web services transactions. In *The 14th International Conference on World Wide Web (WWW '05)*, pages 1130–1131, New York, NY, USA. ACM Press.

Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001). Web Services Description Language (WSDL) 1.1. Technical report, World Wide Web Consortium.

Clement, L., Hately, A., von Riegen, C., and Rogers, T. (2004). UDDI version 3.0.2. Technical report, OASIS.

Dalal, S., Temel, S., Little, M., Potts, M., and Webber, J. (2003). Coordinating Business Transactions on the Web. *IEEE Internet Computing*, 7(1):30–39.

Fantechi, A., Gnesi, S., Lapadula, A., Mazzanti, F., Pugliese, R., and Tiezzi, F. (2009). A logical verification methodology for service-oriented computing. Technical report, Universita' degli Studi di Firenze.

Fauvet, M.-C., Duarte, H., Duman, M., and Benatallah, B. (2005). Handling Transactional Properties in Web Service Composition. In *The 6th 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 Transactions Database Systems*, 8(2):186–213.

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

Greenfield, P., Fekete, A., Jang, J., and Kuo, D. (2003). Compensation is not enough. In *The 7th International Enterprise Distributed Object Computing Conference (EDOC)*, pages 232–239.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H. F., Karmarkar, A., and Lafon, Y. (2007). SOAP version 1.2. Technical report, World Wide Web Consortium.

Henning, M. (2006). The rise and fall of CORBA. *Queue*, 4(5):28–34.

Kelton, W. D., Sadowski, R. P., and Sturrock, D. T. (2004). *Simulation with Arena*. McGraw-Hill: Boston, MA, third edition.

Kilgore, R. A. (2003). Simulation Web services with .Net technologies. In *Proceedings of the Winter Simulation Conference*, volume 1, pages 841–846. IEEE.

Lapadula, A., Pugliese, R., and Tiezzi, F. (2008). Specifying and Analysing SOC Applications with COWS. In *Concurrency, Graphs and Models*, volume 5065 of *Lecture Notes in Computer Science*, pages 701–720. Springer.

Lawrence, K. and Kaler, C. (2006). Web Services Security: SOAP Message Security 1.1 (WS-Security 2004). Technical report, OASIS.

Levy, E., Korth, H. F., and Silberschatz, A. (1991). An optimistic commit protocol for distributed transaction management. *SIGMOD Record*, 20(2):88–97.

Lin, L. and Liu, F. (2005). Compensation with dependency in Web Services composition. In *The International Conference on Next Generation Web Services Practices (NWeSP)*, pages 183–188. IEEE Computer Society.

Little, M. (2003). Web Services transactions: past, present and future. In *XML 2003*.

Little, M. and Freund, T. (2003). A comparison of Web Services transaction protocols. Technical report, IBM Corporation.

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

Mayer, P., Schroeder, A., and Koch, N. (2008). A model-driven approach to service orchestration. In *International Conference on Services Computing*, Honolulu, USA.

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)*.

Milanovic, N., Stantchev, V., Richling, J., and Malek, M. (2003). Towards adaptive and composable services. *Proceedings of the IPSI2003*.

Papazoglou, M. P. (2003). Web Services and Business Transactions. *World Wide Web*, 6(1):49–91.

Paul, D., Henskens, F. A., and Hannaford, M. (2010). Per-request Contracts for Web Services Transactions. In *6th International Conference on Web Information Systems and Technologies (WEBIST-2010)*. INSTICC.

Paul, D., Wallis, M., Henskens, F. A., and Hannaford, M. (2008). Transaction support for interactive Web applications. In *The 4th International Conference of Web Information Systems and Technologies (WEBIST'08)*.

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 *The International Conference on Web Engineering*, Como, Italy. Springer Berlin/Heidelberg.

Tiezzi, F. (2009). *Specification and analysis of Service-Oriented Applications*. PhD thesis, Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze.

Wang, T., Vonk, J., and Grefen, P. W. P. J. (2007). TxQoS: A Contractual Approach for Transaction Management. In *EDOC*, pages 327–338.

Winston, C. K. (1999). An Experience Simulating Web Services. In *CMG Conference*, volume 2, pages 665–669. Computer Measurement Group.

Younas, M., Eaglestone, B., and Holton, R. (2000). A formal treatment of the SACReD protocol for multi-database Web transactions. In *The 11th International Conference on Database and Expert Systems Applications (DEXA'00)*, pages 899–908, London, UK. Springer-Verlag.

Younas, M., Li, Y., and Lo, C.-C. (2006). An Efficient Transaction Commit Protocol for Composite Web Services. In *The 20th International Conference on Advanced Information Networking and Applications (AINA'06)*, volume 1, pages 591–596, Washington DC, USA. IEEE Computer Society.