First published in:

Strengths and Weaknesses of WS-BusinessActivity for Cross-Organizational SOA Applications

Frédéric Wenzel[1,2], Patrick Freudenstein[1], Martin Nussbaumer[1]

[1] Karlsruhe Institute of Technology, Steinbuch Centre for Computing,
76128 Karlsruhe, Germany
wenzel@ira.uni-karlsruhe.de,
{patrick.freudenstein,martin.nussbaumer}@kit.edu

[2] Carnegie Mellon University, Software Engineering Institute,
4500 Fifth Avenue, Pittsburgh, PA 15213, USA

Abstract

In traditional database-driven applications, transactional integrity is a well-established concept. In order to apply these techniques to SOA-based applications, along with the capability to perform long-running business activities, the WS-BusinessActivity standard was developed. This standard does not specifically address cross-organizational use, however, when an enterprise decides to integrate its SOA-based system with their business partners' services, the ability to span long-running business activities across organizational boundaries becomes crucial. This paper describes an experiment performed to assess the suitability of WS-BusinessActivity for cross-organizational use and identifies several strengths and weaknesses that became apparent during its execution.

1 Introduction

In the world of classical, database-driven, applications, transactions are a common concept. For example, banks want to make sure that an account is credited exactly once for a deposit, and of course clients want to be certain that withdrawals from their accounts make it safely into the recipients' hands as well. In this respect, service-oriented architecture (SOA) applications do not differ. But due to the peculiar nature of distributed, service-based systems, transactions have gained new relevance with the rise of the SOA paradigm.

The traditional technique to ensure data consistency in the presence of failures is the use of *atomic transactions*, characterized by the atomicity, consistency, isolation, durability (ACID) properties [7]. However, due to the lack of central data storage, this approach cannot be trivially applied to all actions performed in a distributed, service-based environment. In order to achieve the same transactional properties that are common in client-server applications, the two-phase commit (2PC) approach known from distributed databases [8, pp. 152ff.] has been adapted for SOA applications, in the form of the WS-AtomicTransaction (WS-AT) standard [12].

For long-running business processes, however, atomic transactions turned out to be too strong, in particular because the locking of resources in the first step of the 2PC protocol is unacceptable for extended periods of time [11], and a strict, data-centric *rollback* does not account for non-data-related actions performed by services (for example, sending an email). Thus, for long-running business activities, the WS-BusinessActivity (WS-BA) standard [1, 5] was developed, relaxing the ACID properties by replacing the strict commit/rollback concept with the optimistic notion of *compensation* in the case of error (no matter if this means performing a full rollback, executing a "plan B", or any other action).

Long-running transactions become particularly relevant when an enterprise, having successfully deployed a service-based system to run their internal business processes, considers reaching out beyond their inner boundaries, either to connect autonomous parts of the same enterprise, or to integrate services with their business partners. This is a logical second step for a company, as one of the major reasons for adopting SOA in the first place is to enable interoperability with business partners. With this paper, we aim to share our experience in setting up and evaluating such a system in small scale:

- Section 2 puts the paper into the context of related work.
- After a short overview of WS-BA (Section 3), we describe our experimental design and setup of a transactional, cross-organizational, workflow-based SOA application (Section 4).
- We present the major strengths (Section 5) and weaknesses (Section 6) with both the standard itself and the chosen implementation that became apparent during the execution of the experiment, and which we found to be universally valid and generally applicable.
- For each of the issues, we point out possible solutions to overcome them in the future.

Thus, our research can be useful in two ways: By making possible *users* aware of current limitations as well as helping to further evolve the WS-BusinessActivity *standard* for the use in cross-organizational environments.

2 Related Work

Since the first publication of the WS-AT and WS-BA standards, much related research has emerged, discussing various aspects of service-based transactions using these standards.

Some scholars focus explicitly on the implementation of these standards in a service-based system, pointing out problems of the standards in the process. For instance, [14] first analyzes the WS-Coordination and WS-BA standards, identifying a SOA paradigm violation through the tight coupling of the participants in the transaction. They proceed to describe a prototypical implementation of a transactional middleware designed to divide the business logic from the transaction coordination logic. However, their approach requires the middleware to intercept business messages, effectively prohibiting the system from completely disconnecting the transaction coordinator from the SOA system running the business logic. This problem is related to the limited coordinator flexibility as pointed out in this paper in Section 6.3. [3] further investigated this problem and proposed a "WS-BA-Initiator" extension to the standard in an effort to uncouple initiator and coordinator.

[17] is a master's thesis focusing on the same topic as this paper: cross-organizational SOA transactions. They, however, propose an own transaction processing model for web services, followed by a prototypical implementation of a transactional web-service middleware. Similarly, [15] proposes a new business transaction framework with a focus on Quality of Service (QoS) aspects and e-contracting. In contrast, this paper discusses in its core cross-organizational transactions and a concrete, publicly available and thus widely usable implementation of the WS-BA standard is analyzed.

Other research involving the standards discussed in this thesis includes [13], who compare WS-BA with the long-running transactions support of Business Process Execution Language (WS-BPEL) and propose a modified compensation concept in WS-BPEL, deprecating WS-BA altogether—unlike this paper, which is focused on the applicability of already existing WS-BA implementations in relevant scenarios, not necessarily their deprecation.

The problems of WS-BA identified in this paper therefore add to the list of criticisms researchers have published about the standard, intending to aid in the further evolution of the specification.

3 WS-BusinessActivity Overview

WS-BA employs a coordinator-based pattern following the WS-Coordination standard [4]. The main actors are the coordinator itself, which provides registration, activation and protocol-specific services according to the WS-Coordination standard, the initiator and the participants.

Essentially, the initiator, for example a client application, requests a new transaction context, then invites participant services to join the newly created transaction. The participants do so, and decide if they ("participant completion") or the coordinator ("coordinator completion") will declare when their work for the current business process is completed.

During the course of the transaction, coordinator and participants exchange transaction-related messages to advance the participant through the different states of a transaction's life cycle. For both the *participant completion* and *coordinator completion* protocols, the standard document contains state diagrams [5]. The main focus in this paper lies on the case of failure *after* the completion of the participant's work. In this case, the coordinator calls the compensation actions on all participants.

[5] and [2] provide a more in-depth description of the WS-BA standard and its rationale.

4 Experiment Objective, Methodology, and Setup

In order to investigate the practical applicability of WS-BA, we designed a T-Check experiment. The "T-Check" approach is a method for the context-based evaluation of technologies [10]. The approach involves a two-step process: 1) Formulating qualitative hypotheses based on the context the technology is to be evaluated in, along with the definition of specific criteria that can be used to sustain or refute these hypotheses; 2) The design and implementation

of a simple prototypical solution in order to evaluate the hypotheses against the criteria.

These are the research questions the T-Check was designed to answer:

1. Can an existing implementation of WS-BusinessActivity be used to realize a transactional, inter-organizational workflow?

2. In the case of failure, will the already performed actions in both domains be compensated as expected?

From these questions, the following hypotheses were derived:

1. An instance of a SOA framework, along with its web service stack and implementation of WS-BA, can be leveraged, ideally "off the shelf", in order to create and run a transactional, inter-organizational workflow.

2. Failure of any involved service will lead to compensation actions being executed on all services.

Next we designed a simple, cross-organizational scenario to evaluate these criteria on.

4.1 Scenario

Large parts of the research this paper is based on were performed as master's thesis research at both Karlsruhe Institute of Technology (KIT), Germany, and Carnegie Mellon University (CMU), PA, USA [16]. The scenario picked for the experiment was therefore the example of a workflow for creating a joint master's thesis between these universities, making the process both long-running and cross-organizational.

One part of the process that already spans the two universities' domains, but is still simple enough to serve as a straightforward example, is the KIT Student's work on a viable thesis proposal. After creating an initial draft, the student sends the proposal over to the CMU Advisor, asking them to review the document. The advisor reads it, adding the proposal to his stack of read proposals, then replies to the student if he accepts it or requires further changes. In the case of acceptance, the process ends successfully. If the advisor rejects the proposal, the student modifies his existing paper, and sends it back to the advisor for them to approve or reject (cf. Figure 1).

At any time during this process, either party can also decide to cancel the agreement, for example because they have made other plans. In that case, the advisor removes the student's proposal revisions from their stack, and the student reverts his proposal to its initial state—leaving both parties in the situation they were in before entering the process.

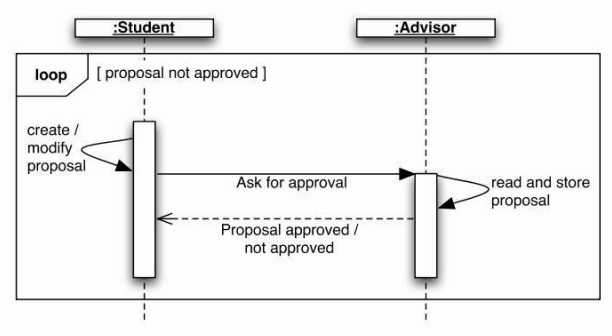

Figure 1. Sequence diagram of the proposal creation scenario.

The scenario chosen for the experiment is intentionally quite simple: It was important to cover a basic, common subset of all cases occurring in real-world applications. Albeit basic, the scenario turned out to be complex enough already to identify the strengths and weaknesses discussed in Sections 5 and 6. Yet, increasing complexity may surface further problems that could not be seen with the simple setup investigated here. It is therefore possible and desirable to iteratively increase the complexity from this starting point on in order to observe the behavior of the system in the process.

4.2 Logical Experiment Setup

This scenario can be described in a SOA-based system. The logical architecture shows how the actors from the scenario are represented in the system, and what actions these entities perform.

There are four central entities involved in the architectural representation of this scenario, as illustrated by Figure 2. First, there are both the advisor service and the student service. They are part of distinct domains (CMU and KIT, respectively), and execute the business logic outlined in the scenario description above. Obviously, in a real-world application, these services would provide the advisor and student with an interface to make their decisions. This was simulated in the experiment by having the services themselves randomly generate the decisions at runtime.

In addition, a third entity is needed, which is called the "workflow controller" in this experiment. Even though it does not play a role in the initial scenario, it acts as the owner of the workflow logic. It initiates the overall business process and drives it forward as illustrated by the scenario's sequence diagram (Figure 1).

As a fourth entity, it is the WS-BA coordinator's task to provide independent transaction services to the other entities and to gracefully compensate the process in the case of

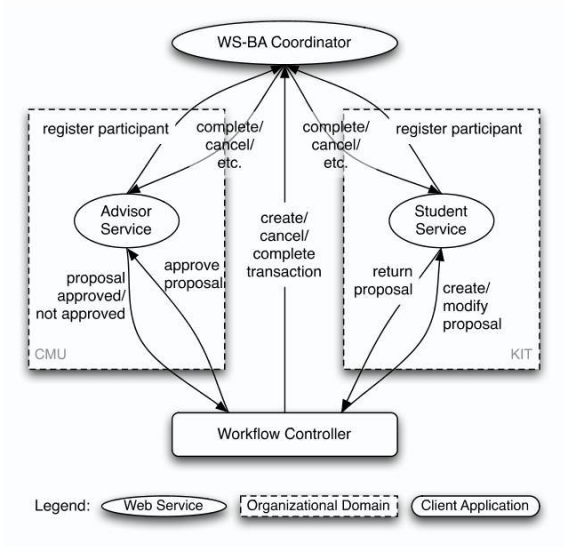

Figure 2. Logical architecture layout of the T-Check experiment.

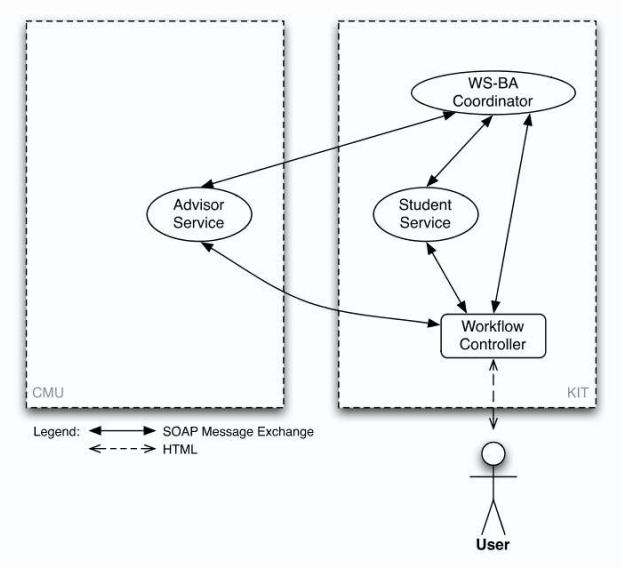

Figure 3. Physical architecture layout of the experiment.

failure, or to make changes persistent in the case of success.

Figure 2 also shows how these four entities interact. The workflow controller initiates the process by creating a new transaction with the WS-BA coordinator. It then asks the student service to create an initial proposal. The student service does that and registers itself as a participant of the transaction before returning the proposal to the controller. The workflow controller takes the proposal and passes it on to the advisor service for approval. The advisor service decides on accepting the proposal after registering itself as part of the transaction as well. If the advisor service refuses the proposal, the controller has the student service modify it, then passes the proposal back to the advisor service—until it is approved, or either of them cancels the transaction. In the case of success, the WS-BA coordinator asks all participants to complete their work and make their changes permanent; in the case of failure, the coordinator asks them to compensate for their changes.

4.3 Physical Architecture, Technologies, and Implementation

The physical architecture of the T-Check experiment describes what physical components the logical entities introduced above were mapped to, and what protocols they adhere to and employ to communicate with each other.

Both the student service and the advisor service were implemented as SOAP web services with persistent background storage as their disposal. The transaction coordinator consisted of a bundle of web services implementing the WS-BusinessActivity 1.1 specification [5]. The workflow controller, finally, was written as a web service client, calling the individual web services according to their WSDL-based service contracts.

In Figure 3, the physical layout of the T-Check experiment can be seen. There were two application servers involved, one hosting the KIT Student service as well as the WS-BA coordinator and the workflow controller; the second one only running the CMU Advisor service. While this seems like the three co-hosted entities are tightly bound, it is worth mentioning that all communication between the individual components happens through SOAP over Hypertext Transfer Protocol (HTTP) messages, therefore upholding the autonomy of the four components as demanded by the logical architecture.

As the two application servers hosting the experiment's components, two instances of the same version of the Java EE-based JBoss Application Server were used. The choice fell on the JBoss project (`http://jboss.org`), because it is both an Open Source solution and a commercial product, offering a high degree of insight and community support, but at the same time providing comparability with the technologies used in actual enterprise scenarios.

On top of the bare application server, the native JBossWS web service stack was used, as well as the JBoss XML Transaction Service (XTS), which contains both client libraries as well as server-side services implementing the WS-AT and WS-BA standards.

Employing these frameworks, the transactional, cross-organizational workflow was realized as described in the physical layout.

4.4 Key Results

The first hypothesis, *"WS-BusinessActivity can be used to establish a transactional, inter-organizational workflow"*, is mostly *sustained*. It was possible to create a workflow-based SOA application and make it transactional, in spite of it crossing the boundaries of two SOA instances. Nonetheless, when implementing the T-Check, it became clear that there are limitations to the currently available technology.

The second hypothesis, *"such a workflow is compensable in the case of failure"*, is *sustained*, but there were some reservations here as well.
On the positive side, our key findings were:

- As an effect of WS-BA being a purely message-based transaction model (Section 5.1), such transactions can easily cross SOA system boundaries.
- The participating services are able to stay stateless in spite of a transaction log being kept, as the WS-BA coordinator handles this completely without the services' involvement (Section 5.2).

In contrast, we identified a few drawbacks:

- The WS-BA coordinator itself represents a single point of failure for the application. What happens if its "plug is pulled" (Section 6.1)?
- A problem with the compensation of the Student service's actions made clear that compensation order is a problem currently not covered by the WS-BA standard (Section 6.2).
- While JBoss' XML Transaction Service fulfilled the requirements of the investigated scenario very well, it became apparent that a more demanding, dynamic setup would have reached the limits of the current framework implementation (Section 6.3).

5 Strengths

Both key strengths of WS-BA that we encountered lie in the standard itself and are thus applicable to all standards compliant WS-BA implementations. However, these advantages do not come for free: Together, they influence two of the three weaknesses talked about in Section 6.

5.1 Message-Based Transactions

All communication within the WS-BA transaction protocol *relies on SOAP messages only*. This property prepares the WS-BA standard inherently for cross-organizational use, as participants in the transactions do not have to physically reside in the same SOA-based system or share anything other than a communication channel.

Just like the participants, the coordinator itself can be outside the SOA instances performing the business logic, and can therefore, for example, be provided by a third, trusted party.

Therefore, WS-BA's message-based transaction model maintains loose coupling even for transactional applications by not binding them tightly to their transaction provider—a strongly desired property for SOA applications [9, p. 46 ff.].

5.2 "Enlist and Forget"

Another advantage of the WS-BA transaction model is the fact that services can *register a participant* with the coordinator, then *forget about it*. The coordinator will call the functions on the participant later when appropriate.

This keeps the participants from maintaining their own transaction log, which they would have to update according to the progress of the transactions they participate in. Instead, it is possible to store all data needed to compensate the actions with the participant when enlisting it in the transaction.

Conceptually, this facilitates statelessness of the service participating in the transaction, as the service does not need to maintain a state for each of the transactions it is a part of, but instead leaves this task up to the coordinator.

6 Weaknesses

Along with its strengths, there are also a number of drawbacks of both the WS-BA standard and the WS-BA implementation used in the experiment. The first two weaknesses lie in the standard itself, and while the third issue is a limitation of the WS-BA framework employed in the T-Check experiment, it is a problem that is generally applicable to all other WS-BA implementations as well.

6.1 "Pulling the Plug"

In this experiment, the completion, cancellation and compensation actions were executed on the WS-BA participants as expected, and it can be assumed that any type of error condition leading to a synchronous error in the application code (i.e., any exception being thrown and properly caught) will lead to the correct behavior.

However, it stays unclear what happens when *connectivity problems* occur, particularly when the coordinator is not local anymore, but can dynamically reside anywhere, as suggested above. The positive notion of "enlist and forget" (Section 5.2) turns into a negative effect when the WS-BA

coordinator "vanishes", as the actions specified by the WS-BA standard are only called by the coordinator itself. Without the coordinator, the participants do not take any further actions.

To mitigate this problem, there needs to be a mechanism that defines what happens if the coordinator becomes unresponsive. We propose the employment of a software component acting as a local "helper agent", which performs graceful degradation in the case of coordinator failure. As a *local* entity, at least one of them is needed per domain participating in the transaction. The specific timeouts and possible other limits would need to be defined in service level agreements, and should be consistent across the participating systems in the same respective transaction. Note that these helper agents remain "quiet observers" during normal operation. They stay up-to-date on the current state of the transactions involving services that they are responsible for, but do not interfere with the WS-BA transaction logic, except if they determine the coordinator to be unresponsive, in which case they gracefully end the affected transactions locally.

6.2 Sequencing Matters

Another issue became apparent when discussing the compensation options for the student service: the question of compensation order. As it turns out, *sequencing matters*.

In the experiment, on each call to the advisor service, it registered an additional participant with the business activity. When an error occurred, this resulted in each of these participants' compensation action being called: once for each iteration of the thesis. Eventually, the advisor's data would reach the state it was in before the start of the workflow, as expected.

But the student service's compensation action was not that easy: As its changes to the thesis proposal document were *incremental* rather than *cumulative* like the advisor service's "logging" of read proposals, compensation had to happen in exactly the *opposite order* of the incremental changes executed on the initial proposal. The WS-BA standard, however, does not guarantee a compensation order, and leaves it completely up to the implementation: Thus, compensation calls can be made in ascending/descending, or arbitrary order, or even all *in parallel*. As a naive solution it would be possible to write code on the participant side to force the participants' compensation actions to be executed in the desired order, for example by passing around some sort of "token" and blocking all compensation actions until they receive the token. But worse than relying on a specific compensation order provided by the coordinator, this would not only violate the autonomy principle by interfering with the coordinator's logic, it would also open the door for unintended side effects, such as deadlocks, when other services behave similarly and suddenly start waiting on each other to "go first", indefinitely.

In this case, it was possible to circumvent this problem. Instead of using one participant per service call, a single participant with coordinator completion was used, as it is *not the student service's decision* if its work in this workflow is done. Instead, the workflow controller decides, based on the advisor service's reply, if it is necessary for the student service to modify the proposal again or not. Finally, the advisor service's WS-BA participant was therefore implemented as described above, but the student service's participant registered for *coordinator completion*, only once for the entire duration of the workflow. Moreover, its changes to the proposal were not made permanent until the end of the workflow, when the coordinator called the student service's *completion* action. At that point the student service made all changes permanent at once. When the business activity was canceled before the student service had been told to complete its work ("cancellation"), the pending changes could therefore simply be discarded. When the activity was cancelled afterwards ("compensation"), the student service was able to compensate the whole bunch of changes to their original state at once, instead of incrementally.

While in this case, reducing the number of participants accessing the same resource to a single one solved the problem, this is not a good, general solution to the sequencing issue: For example, by making all changes temporary and only writing them to persistent storage at the very end, the service is effectively keeping a transaction log of its own, thus negating the advantage of "enlist and forget" (Section 5.2). Also, keeping this record inside the service, spanning multiple invocations of the service, establishes a *session* on the service side and makes the service *stateful*. In SOA-based systems, keeping sessions and states is often not appropriate, as it can lead to problems by tightly coupling services in an otherwise loosely coupled environment.

Additionally, in a real-life scenario, numerous services may modify the same data and may only be able to be compensated for in a specific order. On the one hand, as seen above, forcing a specific compensation order in the participants' code may provoke race conditions because the services know nothing about the other participants' behavior. There is, on the other hand, no way to tell the "omniscient" coordinator that sequencing matters to the service in question. As a result, none of the actors claim responsibility for the sequence of actions, so in some situations compensation may fail for the simple reason that it did not occur in the right order. This is a problem that is currently entirely in the hands of the business developer and represents another obstacle in the creation of wide-spread, transactional workflows. Therefore, it should be handled on the WS-BA standard's level, rather than in the respective application's business logic.

6.3 Limitations of the XML Transaction Service

The JBoss XML Transaction Service used as a WS-BA framework here has, at the time of writing, not been able to be configured per application but only server-wide. That means, all application packages deployed to the same server instance have to use the same coordinator. While using a stand-alone coordinator is possible, dynamically picking one at runtime is not. Statically bound transaction services are a fair limitation if the objective is to connect only a handful of statically known, local services, but it forms a serious obstacle for the creation of dynamically bound, federated applications. It is positive that the inner logic of the WS-BA standard is hidden from the developer by the framework, and it would certainly be an improvement to have the possibility to choose a different coordinator statically per application. However, in order for applications to become truly federated by dynamically binding at runtime to many different services in entirely different SOA instances, the transaction framework will need to become more flexible.

This is a problem that is not limited to the framework used in this experiment. In general, to facilitate dynamic, cross-organizational SOA applications, it needs to be possible to query a service registry, such as a Universal Description Discovery and Integration (UDDI) registry, for any WS-BA coordinator service (JBoss or not), then bind to it dynamically and use its transaction services for the duration of the workflow.

7 Conclusions and Future Work

When integrating their SOA-based infrastructure with business partners, enterprises face the need for proper, service-based transaction management that supports cross-organizational use. In order to determine the suitability of the WS-BusinessActivity standard for this task, we performed an experiment using the T-Check process. During this research, we made several key findings—both positive and negative—which we presented above.

We were able to show that WS-BA is suitable for cross-organizational SOA-based transactions with relaxed ACID properties. Putting the transaction state into the hands of the coordinator supports SOA's statelessness principle, and the message-based standard opens the door for cross-organizational use. However, both the implementation used in the experiment as well as the standard itself have limitations: We pointed out the lack of graceful degradation in the case of coordinator failure, undefined compensation order, and, depending on the implementation, it is impossible to dynamically bind to the WS-BA transaction services at runtime.

For possible *users* of WS-BA in a cross-organizational environment, the standard is therefore most appropriate under the following circumstances:

- when service-based transactional integrity is desired that can both be used inside a single SOA instance as well as across multiple organizational domains
- when sequencing is not relevant for the correctness/feasibility of their services' compensation actions
- when the WS-BA coordinator is *local* to the user's domain and not controlled by a third party, due to undefined behavior in the case of coordinator failure.

From a *research perspective*, future work remains: For example, the undefined behavior in the case of coordinator failure is a serious drawback that needs further research. It is necessary to investigate what precise semantics a solution must have, and if there are other approaches besides the *local agents* we proposed above. Also, the added complexity to the standard should be kept to a minimum.

Our future research interest lies in *compensation sequencing*: In the database world, transaction compensation has traditionally been performed in reverse order [6], just as required in this experiment. However, in large, long-running business activities, parallelization and other optimizations are likely to lead to a significant performance increase that should not be unused unless required otherwise by the application. An approach to the problem should therefore maximize the likelihood of successful compensation in spite of the ACID property relaxations in long-running business activities. Deadlocks should be effectively prevented. At the same time, the additional overhead should be minimal, for example by still allowing for parallelization in cases where sequencing is irrelevant.

Finally, due to the simplicity of the scenario analyzed for this paper, problems occurring in larger applications only have not been investigated. There are questions such as the performance overhead of WS-BA and therefore its scalability that deserve further attention in the context of a more complex experiment.

Acknowledgments

Frédéric Wenzel was supported by a scholarship from the "Landesstiftung Baden-Württemberg" foundation and the "interACT" exchange program between KIT and CMU. The authors would like to thank Grace Lewis, Dennis Smith and Sriram Balasubramaniam from the Software Engineering Institute for their help.

References

[1] L. F. Cabera, G. Copeland, M. Feingold, T. Freund, R. W. Freund, S. Joyce, J. Klein, D. Langworthy, M. Little, F. Leymann, E. Newcomer, D. Orchard, I. Robinson, T. Storey, and S. Thatte. Web Services Business Activity Framework (WS-BusinessActivity). Technical report, Arjuna Techn., BEA Systems, Hitachi, IBM, IONA, Microsoft, November 2005.

[2] T. Erl. *Service-Oriented Architecture : Concepts, Technology, and Design*. Prentice Hall PTR, August 2005.

[3] H. Erven, G. Hicker, C. Huemer, and M. Zaptletal. The Web Services-BusinessActivity-Initiator (WS-BA-I) Protocol: an Extension to the Web Services-BusinessActivity Specification. *icws*, 0:216–224, 2007.

[4] M. Feingold and R. Jeyaraman. Web Services Coordination (WS-Coordination) Version 1.1. `http://docs.oasis-open.org/ws-tx/wscoor/2006/06`, July 2007.

[5] T. Freund and M. Little. Web Services Business Activity (WS-BusinessActivity) Version 1.1. `http://docs.oasis-open.org/ws-tx/wsba/2006/06`, Apr 2007. [Online; accessed 16-Jan-2009].

[6] H. Garcia-Molina and K. Salem. Sagas. *SIGMOD Rec.*, 16(3):249–259, 1987.

[7] J. Gray. The transaction concept: Virtues and limitations (invited paper). In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, pages 144–154. IEEE Computer Society, 1981.

[8] D. Kaye. *Loosely Coupled: The Missing Pieces of Web Services*. RDS Press, 2003.

[9] D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices*. Prentice Hall Ptr, 2004.

[10] G. A. Lewis and L. Wrage. A Process for Context-Based Technology Evaluation. Technical Report CMU/SEI-2005-TN-025, Software Engineering Institute, June 2005.

[11] M. Little. Transactions and web services. *Commun. ACM*, 46(10):49–54, 2003.

[12] M. Little and A. Wilkinson. Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.1. `http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec/wstx-wsat-1.1-spec.html`, Jul 2007.

[13] P. Sauter and I. Melzer. A Comparison of WS-BusinessActivity and BPEL4WS Long-Running Transaction. In P. Müller, R. Gotzhein, and J. B. Schmitt, editors, *KiVS*, Informatik Aktuell, pages 115–125. Springer, 2005.

[14] F. H. Vogt, S. Zambrovski, B. Gruschko, P. Furniss, and A. Green. Implementing Web Service Protocols in SOA: WS-Coordination and WS-BusinessActivity. *E-Commerce Technology Workshops, Seventh IEEE International Conference on*, 0:21–28, 2005.

[15] T. Wang. Towards A Transaction Framework for Contract–Driven, Service–Oriented Business Processes. In *Proceedings of the IBM PhD Student Symposium at ICSOC05*, pages 34–48, 2005.

[16] F. Wenzel. Transaction Management Challenges for Cross-Organizational, Workflow-Based SOA Applications. `http://research.tm.uka.de`, March 2009. Master's Thesis; to be published; Karlsruhe Institute of Technology.

[17] X. Yao and M. B. Dan-Rognlie. Distributed Transaction Management in SOA-based System Integration. Master's thesis, IT University of Kopenhagen, Sep 2007.