

# Providing Decision Capabilities to Coordinators in Distributed Processes

Michael von Riegen, Martin Husemann, Norbert Ritter  
Distributed Systems and Information Systems  
University of Hamburg  
Vogt-Kölln-Straße 30, 22527 Hamburg, Germany  
{riegen,husemann,ritter}@informatik.uni-hamburg.de

**Abstract**—Current approaches to transactional support of distributed processes in service-oriented environments are limited to scenarios where the participant initiating the process maintains a controlling position throughout the lifetime of the process. This constraint impedes support of complex processes where participants may only possess limited local views on the overall process. In this paper, we present a framework to extend WS-BusinessActivity that strengthens the role of the coordinator. We develop a set of rules for deciding on the confirmation or cancellation of participants’ work and introduce protocol extensions for monitoring the progress of a process, allowing coordinators to initiate the completion of processes largely autonomously. We discuss motivating examples as well as existing approaches and evaluate our rule set against various service interaction patterns.

## I. INTRODUCTION

*Service-Oriented Architectures* (SOA) are a popular paradigm to implement loosely-coupled distributed environments [1]. Within these environments, participants can overcome heterogeneity by communicating through standardized, implementation independent interfaces. Application logic is typically not located in monolithic programs, but distributed among the participants of processes. Web Services are a typical implementation of the SOA paradigm, and services are more and more used to support long-running business processes. This trend runs alongside with a shift from merely considering simple interaction behavior of services, i. e., request-response interaction patterns manifested in standards such as SOAP and WSDL, towards conversational services that engage in long-running conversations with other services.

Transactional activity control plays an important role for such processes as it preserves the consistency and provides failure recovery. Among the published specifications for Web Service transactions, WS-Coordination [2], BTP [3] and WS-CAF [4] are the most prominent ones. Several comparisons and analyses have shown that these specifications provide mostly equal functionality, with the fundamental approaches based on specifications such as X/Open DTP [5] and the well-known 2PC protocol [6].

All of the specifications assume that a transaction has an initiator and that this initiator is also the one who is able to decide on the closure of a transaction. This viewpoint is appropriate for service orchestrations where a centralized

engine controls the message and data flows. In contrast to centralized service orchestrations, service choreographies describe interactions from a global point of view, i. e., from the perspective of an ideal observer who is able to see all interactions and their flow and data dependencies [7]. In such processes where many participants work together in order to achieve a common goal, the initiator of a process is not always the one who is able to decide whether to commit or cancel a transaction. In order to support such scenarios, the control of the transactional coordination is intentionally left undefined in the prevalent specifications.

In this paper, we present our work on transactional activity control for scenarios where the initiator is not the one demarcating the transaction. Section II gives motivating examples and considers weaknesses of current approaches as well as open questions. Section III discusses related work for transactional activity control in the Web services domain. Our extension of WS-BusinessActivity is presented in Section IV while Section V gives a conclusion and an outlook on future work.

## II. MOTIVATING EXAMPLES AND ANALYSIS

In this section we introduce two different scenarios and analyze them regarding the usage and suitability of WS-Coordination (WS-C), WS-AtomicTransaction (WS-AT) and WS-BusinessActivity (WS-BA).

### A. Buyer-Supplier Scenarios

The best known examples for Web service transactions are scenarios where a traveler wants to book a travel via a travel agency. The agency provides services for booking hotels, flights and cars. A travel consists of a hotel reservation, a round-trip flight booking and a rental car reservation. The traveler strives for minimal costs and will therefore combine the cheapest individual offers into his actual booking while turning down more expensive offers.

Figure 1 displays a sample WS-Coordination message exchange. This example shows that the travel agency needs to instruct the coordinator whether to commit Airline A or Airline B. WS-AtomicTransaction specifies the *Completion Protocol*, allowing the travel agency to trigger the closure of the transaction at the coordinator [8]. Both airlines would

then confirm their reservations. The required behavior in the scenario above, however, is that one airline confirms its reservation and one discards it. Such *Mixed Outcome* scenarios are not supported by WS-AT, but only by the WS-BusinessActivity specification [9]. WS-BA does not contain a counterpart to WS-AT's Completion Protocol though; the communication between the travel agency and the coordinator is thus unspecified.

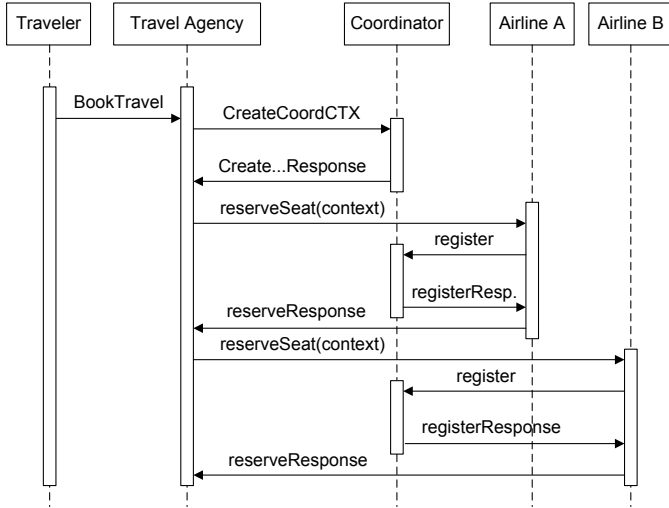


Fig. 1. A sample WS-Coordination message interaction

Even this basic example shows that although coordination specifications at first separate transactional coordination from application-specific functionality, non-trivial scenarios require application-specific knowledge for coordination decisions. An application-neutral, i.e., not specifically hard-coded, coordinator cannot decide which participants are to commit or to cancel a transaction. The necessary knowledge resides with the respective application, which usually forms the initiating participant of a transaction or, more broadly speaking, the *initiator* of a distributed process. We identified two concepts to attach an application to an application-neutral coordinator:

- The *proxy* concept: The coordinator only acts as a proxy, i.e., as a messaging hub, between the initiator and the other participants. It forwards decisions made by the initiator, assures that every participant adheres to these decisions, and reports the result of the coordination back to the initiator. The coordinator is explicitly steered by the initiator via a control protocol. WS-AT's Completion Protocol is an implementation of the proxy concept.
- The *callback* concept: The coordinator handles the coordination basically on its own. In case of application-specific decisions, it contacts the initiator via a callback interface to query its decision, which it then enforces on the participants. This approach allows for a greater degree of decoupling of the initiator from the coordination than the proxy concept since the initiator does not need to actively push the process along.

With both concepts, the coordinator is still dependent on

the initiator concerning the contents of decisions. The callback concept thus mainly offers technical advantages over the proxy concept. Several research projects have implemented the proxy concept for WS-BA (cf. Section III), but to date there is no implementation of the callback concept.

### B. Europol/Eurojust Scenario

Two agencies - Europol and Eurojust - have been founded to help the EU Member States co-operate in the fight against cross-border organized crime. Co-operation in criminal matters is a subject dealt within the third pillar of the EU (Title VI of the Treaty of the European Union). Eurojust stands for the European Judicial Cooperation Unit whereas Europol refers to the European Police Office. Europol and Eurojust carry out very specific tasks in the context of dialogue, mutual assistance, joint efforts and co-operation between the police, customs, immigration services and justice departments of the EU member states.

An exemplary interaction between Europol and Eurojust and their subsidiaries may proceed as follows: A Spanish police officer finds cocaine in the harbor of Malaga during a routine check on a ship which has loaded coffee. The container came from Caracas, Venezuela, and was supposed to be transported to Antwerp, Belgium. Several persons are taken into custody in Malaga. Europol and Eurojust have to work together with the member states. The local police officer in this case is only the one initiating the process, but is neither the one who is able to decide which action has to be taken next nor - in the sense of a transaction - is he able to decide what future steps in the process should be canceled or completed.

Neither WS-AT's Completion Protocol nor its adaptations to WS-BA do specify which participants are entitled to register for completion: The process initiator is implicitly regarded as the controlling participant, intermingling the *role* of being the initiator with the *privilege* to trigger the closure of the process. WS-AT's behavior is undefined for cases where more than one participant registers; other approaches do not allow alternative or multiple controlling participants at all. In order to support scenarios as the one described above, at least an explicit management of the *demarcation privilege* is required. However, in complex scenarios it may be difficult to even identify a participant with adequate knowledge of the overall process to reasonably endow it with the demarcation privilege.

We therefore propose a coordination framework that leverages the central position of the coordinator to concentrate a greater share of the overall coordination logic there and in turn reduce the coordination-related complexity on the participants. This way, individual participants are not required to have a complete view of the process, but only a local view of their involvement and the respective conditions, which they share with the coordinator. The coordinator aggregates the local views to a global process state and can decide autonomously when to trigger the closure. This approach, although relying on a centralized coordination, especially supports *choreographies* of loosely-coupled participants.

### III. RELATED WORK

The prevalent role of WS-Coordination, WS-CAF and BTP has been identified by many authors who presented analyses, comparisons and enhancements of those specifications [10], [11], [12], [13]. The enhancements mostly focus on the role of the coordinator and the placement of process control in complex business processes. A common feature of the various approaches is the placement of process control on one distinct participant. They define control protocols analog to the *Completion Protocol* of WS-AtomicTransaction. This has the coordinator act as a mere proxy of the controlling participant and makes the whole process dependent on this participant. The role of the controlling participant is usually implicitly assigned to the process initiator.

Monsieur, Snoeck and Lemahieu propose to integrate transactional coordination into an event notification architecture, using *Model Driven Architecture* concepts to map abstract business events to concrete logic, but do not discuss such a mapping or the actual procedure of the coordination itself [14].

Leymann and Pottinger identify a lack of possibilities to query the state of a coordination in WS-Coordination [11]. They note that case-specific application logic is required and propose to place this logic in a "protocol-specific service" which is coupled with a generic coordinator to form an application-specific coordinator. They also briefly mention coordination rules and parameterization of coordinators by clients in the context of combinatorial auctions, but do not relate these concepts to transaction management.

Vogt et al. aim at separating the coordination from the business logic of distributed processes both on the sides of the process initiator and the ordinary participants. They propose an extended implementation of WS-Coordination which bundles the coordinator and a newly introduced *transactor* in a transactional middleware component [15]. However, while their approach does decouple the initiator from the concrete WS-Coordination protocols, the decision on committing or canceling participants' work remains solely with the initiator which controls the transactor through detailed *Confirm* and *Cancel* messages. The architecture thus effectively presents a protocol defining the technical communication between initiator and coordinator, but does not provide a decoupled coordination with respect to the contents and the logics of coordination decisions.

Another approach to such a protocol has been proposed by Erven et al. [16]. They emphasize the need to free coordinators from application-specific logic to enable the provision of common coordination services, which offer benefits for distributed processes between heterogeneous participants. Their *Web Services-BusinessActivity-Initiator Protocol* allows applications to control WS-Coordination-compliant coordinator services and addresses several aspects left undefined by the WS-Coordination specification. While providing significant improvements for the practical use of WS-Coordination, it is specifically designed to concentrate process control on the initiator and consequently offers no functionality towards

decoupled or let alone autonomous coordination.

### IV. A FRAMEWORK FOR THE COORDINATION OF TRANSACTIONAL ACTIVITIES

There are two fundamental aspects to the transactional coordination of distributed processes: First, *when to end the process*, i. e., when to trigger the *completion* phase that leads to the commitment of work results and a consistent, orderly outcome (the "when"). Second, *which participants are to commit* and which are to cancel their work (the "who"). Within WS-AT, both aspects are clearly defined through the Completion Protocol. When the closure of the process is triggered by the controlling participant, the coordinator executes a two-phase-commit protocol among the participants so that all the participants must commit their work for the process to come to a "successful" end.

The intended coordination type for long-running, complex processes is WS-BA. There, a participant can either autonomously report completion of its local work to the coordinator and thus signal its waiting for a command to commit or discard the results, or the coordinator orders a participant to complete its local work and enter a state waiting for the final commit/discard command. Both variants take individually place between the coordinator and a single participant; there is no defined time at which all participants would enter completion. This procedure does not only leave both aspects of transactional coordination mentioned above undefined, but it also intermingles them: In business activities, not all participants need to commit their work for the process to reach a "successful" outcome. The closure of the process can thus be triggered as soon as all the *necessary* participants have reached a *completed* state. Basic WS-BA, however, does not define how the coordinator can decide to order a participant towards completion nor how the coordinator can assess which participants are necessary for a successful outcome, which effectively requires the coordinator to wait for *all* participants to report completion on their own.

Existing advanced solutions approach both aspects of transactional coordination with control protocols adopted from the Completion Protocol of WS-AT. Through these protocols, the coordinator is explicitly told which participants are to commit or to discard their work and when the respective commands are to be sent. In contrast to this, we propose a framework where the coordinator itself can decide when to close the process and which participants are to commit their work. To this end, the coordinator needs to keep track of the participants' states and the progress of the overall process.

#### A. Maintaining Participant States

The overall set of participants  $P$  is divided into a *complete set*  $C$  and a *cancel set*  $D$ , meaning that at the closure of the process, members of  $C$  will have to commit their work and members of  $D$  will have to discard their work for the process to reach a successful outcome. These sets thus hold the *potential closure behavior* of the participants based on their current states. They are maintained by the coordinator according to

the progress of the process so that the current belonging of each participant is always known. In this way, the "when" and "who" aspects of coordination are effectively decoupled. From a technical point of view, the closure of the process can be initiated at any time; the coordinator has the necessary information to order all participants towards committing or canceling their work.

In many cases, the coordinator can autonomously assign participants to the complete or cancel set in an application-neutral way, based on the interactions between participants, which usually take place as service invocations. Nevertheless, participants need to make application-specific decisions themselves and report the results to the coordinator. Only in this way the coordinator can remain application-neutral: Leaving application-specific decisions with the participants allows encapsulating complex application logic within the application while outsourcing the technical aspects of the associated transactional coordination onto the coordinator.

We identified a set of fundamental rules which allows assigning participants to the complete or cancel set. A participant  $B$  is assigned to the complete set of a coordination context  $Z$  (1) if it is invoked by a participant  $A$  which is a member of the complete set or (2) if  $B$  is invoked by  $A$  as a member of a *choice set*,  $A$  is a member of the complete set and  $B$  is a *chosen member* of the choice set:

$$\forall A \forall B \forall Z (invocation(A, B, Z) \wedge completion(A, Z) \wedge \neg(partOfChoice(B))) \rightarrow completion(B, Z) \quad (1)$$

$$\forall A \forall B \forall Z (invocation(A, B, Z) \wedge completion(A, Z) \wedge partOfChoice(B) \wedge choose(B)) \rightarrow completion(B, Z) \quad (2)$$

Accordingly, a participant  $B$  is assigned to the cancel set of a coordination context  $Z$  (3) if it is invoked by a participant  $A$  which is a member of the cancel set or (4) if  $B$  is invoked by  $A$  as a member of a choice set and  $B$  is not a chosen member of the choice set:

$$\forall A \forall B \forall Z (invocation(A, B, Z) \wedge cancel(A, Z)) \rightarrow cancel(B, Z) \quad (3)$$

$$\forall A \forall B \forall Z (invocation(A, B, Z) \wedge partOfChoice(B) \wedge \neg(choose(B))) \rightarrow cancel(B, Z) \quad (4)$$

For regular service invocations, if a service is invoked within a coordination context and the invoker is a member of the complete set, the invoked service is assigned to the complete set as well (Rule 1). This is the most common case: The initiator of a process is usually interested in a consistent outcome and therefore invokes services within a newly created coordination context. If these services invoke further services on their own, they usually need to pass on the coordination context to ensure consistency of these sub-invocations. Consequently, an *invocation tree* of services emerges in which all participants join the original coordination

context created by the initiator and all participants are assigned to the complete set. Figure 2 shows a sample invocation tree.

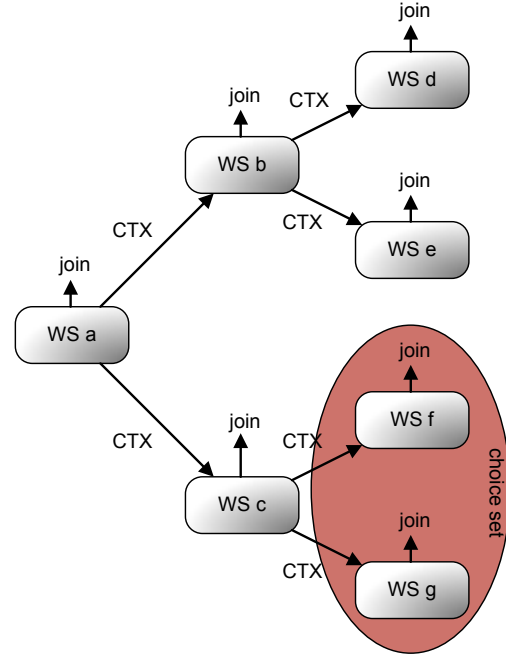


Fig. 2. Sample Web service invocation tree

A participant which itself is not a member of the complete set for some reason (and thus is a member of the cancel set) is usually a passivated participant which only remains in the process to ensure an orderly closure. Invoking further services therefore makes little sense for such a participant. If it does anyway, the invoked service is assigned to the cancel set as well (Rule 3).

In summary, the complete set or cancel set membership of a participant is propagated to any services invoked by this participant. This behavior is particularly important when a participant is moved from the complete set to the cancel set, e.g. because of an application-specific decision: In this case, all participants in the invocation sub-tree of this participant are moved to the cancel set as well (Rule 3).

*Choice sets* are used to model conditional completion, i.e., cases when groups of services are invoked in parallel and only a subset of the work of these services will be committed. A typical example of such cases are buyer-supplier scenarios as outlined in Section II: A number of services is invoked with a request for an offer, but only the cheapest offer is confirmed while the other offers are turned down. In general, choice sets are not limited to "1 out of n" selections, but they can also represent "m out of n" selections. Since the selection criteria can be highly application-specific and dependent on the contents of messages exchanged between the participants, they cannot feasibly be integrated into an application-neutral coordinator. Participants need to actively inform the coordinator about choice sets and their members. These details are piggybacked with existing WS-Coordination

messages: Invoked services are informed about their being part of a choice set via the `CoordinationContext` with the regular invocation so that they can pass on this information when registering with the coordinator (corresponding to the `partOfChoice()` predicate of Rules 2 and 4). Members of choice sets are at first assigned to the complete set. When a participant comes to a decision on which of the members of a choice set to complete and which to cancel, it reports this decision to the coordinator via a newly introduced `Choose` message in the WS-BA coordination protocol (corresponding to the `choose()` predicate of Rules 2 and 4). The coordinator then moves the non-chosen members to the cancel set.

Figure 2 contains an example of a choice set created by Web service  $c$  of Web services  $f$  and  $g$ . If  $c$  chooses participant  $f$  for completion, participant  $g$  will implicitly be assigned to the cancel set.

### B. State Maintenance Rules and Interaction Patterns

To illustrate the feasibility of our participant state maintenance rules, we analyzed their application to various service interaction patterns as categorized by Barros, Dumas and ter Hofstede [17].

#### • one-way or round trip bilateral interactions

- *send*: If a participant is invoked within a coordination context, it joins this context. It is then assigned to the complete or cancel set along with its invoker (R1, R2, R3) unless it is a non-chosen member of a choice set (R4).
- *receive*: The unsolicited reception of a message allows the recipient to choose whether to act on it. If it takes action and joins the coordination context, the rules apply as in the *send* pattern.
- *send/receive*: For a combined send/receive interaction, the rules apply as in the *send* pattern.

#### • single transmission multilateral interactions

- *racing incoming messages*: This pattern can be reduced to *receive*; the rules apply accordingly.
- *one-to-many send*: This pattern can be reduced to a set of *send* interactions.
- *one-from-many receive*: This pattern can be reduced to a set of *receive* interactions.
- *one-to-many send/receive*: This pattern can be reduced to a set of *send/receive* interactions.

#### • multi-transmission interactions

- *multi-responses*: This pattern can be reduced to *send/receive*. The invocation tree is independent of the number of messages sent between two participants.
- *contingent requests*: This pattern can be reduced to a set of *send/receive* interactions between an invoker and members of a choice set. The first invoked service to respond is assigned to the complete set, the others are assigned to the cancel set.
- *atomic-multicast notification*: This pattern can be reduced to a set of *send/receive* interactions between

an invoker and members of a choice set. When the required number of acceptance responses is reached, the remaining invoked services are assigned to the cancel set.

#### • routing patterns

- *request with referral*: This pattern is a combination of *send* and *receive*.
- *relayed request*: This pattern is a combination of *send*, *receive*, and *send/receive*. The relaying participant is not forced to join the coordination context, but it is subject to the rules if it does.
- *dynamic routing*: This pattern can be reduced to a set of *send/receive* interactions, with choice sets if necessary.

The analysis shows that in spite of their diversity concerning the contents, the various service interaction patterns can be reduced to few combinations of technical *send* and *receive* interactions. Our fundamental participant state maintenance rules support all of these interactions. We implemented the rules in Prolog, employing a Prolog interpreter as a decision engine in our framework. Practical experiments confirmed the feasibility of this approach.

### C. Maintaining Process Progress

In order to determine a time to trigger the closure of a process, the coordinator needs a complete view on the progress of the process. When there are no more interactions between the participants, no more new services are invoked and included into the coordination context, and all decisions on choice sets have been made, the completion phase can be initiated. Participants can finish possible remaining local work during the *completing* state according to WS-BA.

The global view on the process is maintained via the invocation tree mentioned above. A newly joining participant is required to report to the coordinator the identifier of the participant that invoked it and how many further services it will invoke. We implemented a participant identity management framework for WS-Coordination which allows piggybacking these details with existing WS-Coordination messages. The coordinator issues identifiers to participants upon registration via the `RegisterResponse` message. Every further message sent by a participant then contains its identifier in an extended `CoordinationContext`. If a participant is invoked by another participant, it includes the invoker's identifier with the `Register` message. Registering without an invoker identifier thus makes a participant a root node in an invocation tree, which is usually only the case for the process initiator. The `Register` message also contains the number of further services a participant will invoke. If a participant does not invoke any further services, it is a *stable leaf* in the invocation tree. In contrast, a participant who will invoke other services is a *temporary leaf* in the invocation tree.

Starting from the initiator, participant interactions result in a hierarchy of service invocations. The coordinator is aware of the current extent of the invocation tree and of

outstanding service invocations. When all of the leaf nodes of the invocation tree are stable leaves, the invocation tree has reached a stable maximum extent. Possible remaining active interactions between participants are usually short-lived and can be finished during the *completing* state just as local work on participants. The coordinator thus only needs to wait for the various decisions on choice sets before it can autonomously initiate the completion phase. No participant is required to possess a global view of the process; the individual local invocations and choice set decisions are only aggregated at the coordinator.

## V. CONCLUSION AND OUTLOOK

In this paper we analyzed current approaches to transactional support of distributed processes in service-oriented environments. A discussion of exemplary use cases showed that existing approaches are weakened by their limitation to scenarios where the participant initiating the process maintains a controlling position throughout the lifetime of the process. In addition, the WS-Coordination and WS-BusinessActivity specifications leave several aspects and protocol details undefined, inhibiting interoperability of existing implementations and extensions. This concerns for example the closure of business transactions and the identity management of participants.

Addressing these drawbacks, we introduced our approach to strengthen the decision capabilities of coordinators for transactions compliant to WS-BusinessActivity. We presented and analyzed predicate rules for assigning participants to the cancel or complete set of a process. The practical implementation of these rules lead to several minor extensions of WS-Coordination, for example an extended Register message containing information about the invoker and intended sub-invocations of a participant registering for a transaction, and WS-BusinessActivity, where an extra message indicating which participants of a choice set are to commit or cancel their work was introduced. An analysis of the predicate rules with regard to common interaction patterns showed that the rules can be universally applied. Our implementation of a coordinator uses a Prolog interpreter to make decisions according to the predicate rules. With these rules, coordinators are enabled to decide nearly autonomously on the completion of processes and the confirmation or cancellation of participants' work; no external control protocol is needed. Details of the implementation as well as support of coordinator interposition and subtransactions had to be omitted due to space restrictions.

The rules presented in this paper are based on the fundamental assumption that all participants which receive a coordination context actually join the transaction and that all participants are *vital* for the process, i.e., participants cannot leave the process on their own decision before it is completed. In the near future, we intend to investigate the support of varying grades of participant vitality and its effects on transaction management. We expect to see greater flexibility and thus a better support of dynamic processes. Furthermore, we will investigate the possibilities of implementing a completely autonomous coordinator which can come to choice set

decisions on its own. Such a coordinator could be integrated into an Enterprise Service Bus framework, allowing process participants to completely outsource transaction management into the communication infrastructure and creating a *transactional* Enterprise Service Bus.

## REFERENCES

- [1] M. P. Papazoglou and W.-J. van den Heuvel, "Service oriented architectures: approaches, technologies and research issues," *VLDB J.*, vol. 16, no. 3, pp. 389–415, Jul 2007.
- [2] M. Feingold and R. Jeyaraman, "Web Services Coordination (WS-Coordination) Version 1.1," Apr 2007. [Online]. Available: <http://docs.oasis-open.org/ws-tx/wscoord/2006/06>
- [3] P. Furniss, S. Dalal, T. Fletcher *et al.*, *Business Transaction Protocol (BTP 1.1)*, 2004. [Online]. Available: [http://www.oasis-open.org/committees/download.php/9836/business\\_transaction-btp-1.1-spec-wd-04.pdf](http://www.oasis-open.org/committees/download.php/9836/business_transaction-btp-1.1-spec-wd-04.pdf)
- [4] D. Bunting, M. Chapman, O. Hurley *et al.*, *Web Services Composite Application Framework (WS-CAF) Ver 1.0*, Jul 2003. [Online]. Available: <http://developers.sun.com/techtopics/webservices/wscaf/primer.pdf>
- [5] *Distributed Transaction Processing: Reference Model, Version 3*, The Open Group, 1993. [Online]. Available: <http://www.opengroup.org/bookstore/catalog/g504.htm>
- [6] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.
- [7] C. Peltz, "Web Services Orchestration and Choreography," *IEEE Computer*, vol. 36, no. 10, pp. 46–52, Oct 2003.
- [8] M. Little and A. Wilkinson, "Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.1," Apr 2007. [Online]. Available: <http://docs.oasis-open.org/ws-tx/wsata/2006/06>
- [9] T. Freund and M. Little, "Web Services Business Activity (WS-BusinessActivity) Version 1.1," Apr 2007. [Online]. Available: <http://docs.oasis-open.org/ws-tx/wsba/2006/06>
- [10] M. Husemann, M. von Riegen, and N. Ritter, "Transactional Coordination of Dynamic Processes in Service-Oriented Environments," in *ICWS*, 2007, pp. 1024–1031.
- [11] F. Leymann and S. Pottinger, "Rethinking the Coordination Models of WS-Coordination and WS-CF," in *Third European Conference on Web Services (ECOWS 2005), 14-16 November 2005, Växjö, Sweden*, Nov 2005, pp. 160–169.
- [12] M. Little and T. Freund, "A comparison of Web services transaction protocols (A comparative analysis of WS-C/WS-Tx and OASIS BTP)," Oct 2003. [Online]. Available: <http://www-106.ibm.com/developerworks/webservices/library/ws-comproto/>
- [13] T. Vetter, "Anpassung und Implementierung verschiedener Transaktionsprotokolle auf WS-Coordination." Master's thesis, Institut für Architektur von Anwendungssystemen - Universität Stuttgart, 2006.
- [14] G. Monsieur, M. Snoeck, and W. Lemahieu, "Coordinated Web Services Orchestration," in *2007 IEEE International Conference on Web Services (ICWS 2007), July 9-13, 2007, Salt Lake City, Utah, USA*, 2007, pp. 775–783.
- [15] F. H. Vogt, S. Zambrovski, B. Gruschko, P. Furniss, and A. Green, "Implementing Web Service Protocols in SOA: WS-Coordination and WS-BusinessActivity," in *7th IEEE International Conference on E-Commerce Technology Workshops (CEC 2005 Workshops), 19 July 2005, München, Germany*, Jul 2005, pp. 21–28.
- [16] H. Erven, G. Hicker, C. Huemer, and M. Zapletal, "The Web Services-BusinessActivity-Initiator (WS-BA-I) Protocol: an Extension to the Web Services-BusinessActivity Specification," in *2007 IEEE International Conference on Web Services (ICWS 2007), July 9-13, 2007, Salt Lake City, Utah, USA*, 2007, pp. 216–224.
- [17] A. P. Barros, M. Dumas, and A. H. M. ter Hofstede, "Service Interaction Patterns," in *3rd Int. Conf. on Business Process Management, Nancy, France*, 2005, pp. 302–318.