

# Adaptable Transaction Processing in the Web Services Domain

Yan Wang and Weihai Yu

Department of Computer Science  
University of Tromsø

{wang, weihai}@cs.uit.no

## Abstract

The advent of Web Services technologies both demands interoperability and adaptability of various transaction processing models, and provides a solid basis for this goal. In this paper we first give an overview of the problem domain and analyze the state-of-the-art Web transaction protocols, the Business Transaction Protocol (BTP) and Web Services Coordination and Transactions (WS-C/T), and then we propose an *Adaptive Transaction Framework* based on extensions to the CORBA Activity Service as a principled way to allow reliable and adaptable Web Services computing. Detailed examples are given to illustrate the applicability of our framework to real world Web Services transactions.

## 1 Introduction

Web services are self-contained, modular business process applications that are based on industry standard technologies such as SOAP [1], WSDL [2] and UDDI [3], for services communication, description and advertising respectively. Web services provide a means for different organizations to connect their applications with one another to conduct business across a network in a platform and language independent manner. It is predicted by Gartner Group that 75% of Web services use will be to integrate new and existing applications [18]. Web services are highly suited to integration, both inside and outside the organization. They offer a platform- and technology-agnostic vehicle for creating interoperability between systems that were never designed to work together.

The fact that Web Services are specifically about fostering systems interoperability projects some interesting issues on the heterogeneity problem of the vast variety of distributed transaction models. Transaction processing plays a key role in guaranteeing reliable and consistent coordination of operations across Web Services. The Web Services platform not only requires the same coordination behavior provided by a traditional ACID (Atomicity, Consistency, Isolation and Durability) transaction mechanism to control the operations and outcome of an application, but also requires the capability to handle the coordination of processing outcomes and results from multiple services in a more flexible manner. Traditional ACID transactions are most suitably viewed as *short-lived* entities, performing stable state changes to the system. Different transaction processing strategies have been proposed concerning concurrency control,

recovery control and distributed transaction management. However, heterogeneity in these mechanisms makes interoperability between different transaction processing systems very hard, if not impossible [4]. On the other hand, in the Web Services domain, many collaborative business process management systems support complex, *long-running* processes, where holding on to resources for a long time would be unacceptable. Furthermore, undoing tasks that have already completed may be necessary in order to affect recovery, or to choose another acceptable execution path in the process. These requirements lead to the relaxation of the ACID properties, and new mechanisms like compensation to be adopted. Indeed, long before Web Services appear in the literature, numerous researches have already been working on developing specific *extended transaction models* that meet the need for relaxed ACID properties [e.g., 5, 6, 7, 8]. However, most of the proposed extended models have not found any widespread usage, and even fewer are being used in a commercial Transaction Processing (TP) monitor. Again, one important reason for this is the lack of flexibility [9], in that a fixed transaction model is not sufficient for all applications, and to hardwire a specific extension mechanism would be inappropriate. In fact, most TP monitors are monolithic in structure and difficult to extend. Ideally, it would be highly desirable to construct an *adaptable* TP system architecture that can adopt different transaction models and properties seamlessly according to the various and ever-changing application needs. The architecture should allow adaptability to be achieved in a *principled* manner rather than ad hoc, both at configuration phase when the TP system starts up and during its run-time. Equally important is its ability to maintain the correctness and integrity of the system when different transaction models and properties in its sub-systems may possibly lead to conflicts.

The Web Services approach provides a basis for this goal, in that it offers an attractive set of technologies by which existing legacy TP systems can be wrapped up as Web Services and made available for integration with other systems within an organization or even cross-enterprises. Web Services architecture is deliberately not prescriptive about what happens behind service implementations, it only concerns with the transfer of structured data between parties. Thus it offers a possible promising solution to the interoperability issue between various transaction models and behaviors. Recently, two competing specifications regarding Web Services transactions have been proposed: OASIS Business Transaction Protocol (BTP) [10], and the Web Services Coordination and Transactions (WS-C/T) protocol [11]. Both attempt to address the problems of running transactions with Web Services. However, we consider these specifications not flexible and general enough to achieve the goal of interoperability and adaptability. We propose an *Adaptive Transaction Framework* based on extensions to the CORBA Activity Service [12] and some of the ideas presented in the above-mentioned Web Services transaction protocols.

The rest of this paper is organized as follows. We present the two available Web Services transaction protocols in Section 2. Section 3 illustrates our adaptable TP processing architecture with several practical examples. In Section 4, a review of related work is given. Finally, Section 5 provides conclusion with a summary and directions of future work.

## 2 Web Services Transaction Protocols

### 2.1 The Business Transaction Protocol (BTP)

OASIS Business Transaction Protocol (BTP) is a specification aimed at business-to-business transactions in loosely-coupled domains such as Web Services. It was worked out by a consortium of companies including HP, Oracle and BEA in April 2002. It was the first cross-industry attempt to produce an XML standard for B2B transactions. It is designed to support applications which are disparate in time, location and administration, and thus require transactional support beyond classical ACID transactions.

BTP defines two *extended transaction* types, namely, *atoms* and *cohesions*, both of which mandate a two-phase completion protocol to ensure atomicity between sub-sets of participants. In the first phase (prepare phase), any participant is expected to make durable the state changes that occurred within the scope of the transaction. During the second phase, these changes can either be undone (cancelled) or confirmed if consensus has been achieved. Although BTP uses a two-phase protocol to ensure that participants and coordinators agree on the outcome of the transaction, it is not ACID. BTP neither requires nor expects the rigid locking of resources that is needed for ACID guarantees.

BTP *atoms* are similar to transactions in tightly-coupled systems, with the exception that the isolation property is relaxed. The effects of interaction may be externally visible before the commit phase. One *atom* coordinator and zero or more sub-coordinators coordinate a transaction. Each manages one or more participants, who act on behalf of services to either accept or reject the work done by the service within the scope of the atom. In addition, the outcome of an *atom* is atomic, such that all of the participants will either confirm or cancel. BTP *cohesions*, on the other hand, in addition to the relaxation of isolation property, may deliver different termination outcomes to its participants, such that some will confirm while the remainder will cancel. Consistency is determined by agreement and interaction between the client (initiator) and the coordinator. *Cohesions* are therefore more general and actually subsume *atoms*.

There are several optimizations to the basic BTP protocol:

- Participant resignation  
This can be the equivalence of read-only optimization in the traditional two-phase commit protocol, or it may happen when a transaction is split into multiple ones [20]. A participant can resign from an *atom* or *cohesion* it was enrolled in. Resignation can occur at any time up to the point where the participant has prepared, and is used by the participant to indicate that it no longer has an interest in the outcome of the transaction.
- Autonomous participant decisions  
Equivalent to traditional heuristic decisions, BTP allows participants to make unilateral decisions concerning whether it will confirm or cancel state changes.
- Carrier optimizations  
Typically a participant is enlisted with a BTP transaction when a service invocation occurs. When the service request completes, the response is sent back

- to the initiator of the request. In some circumstances, it may be possible to compound many of the above messages into a one-shot message.
- One-phase  
If an *atom* or *cohesion* coordinator has only a single participant when it is told to confirm, it can tell the participant to confirm without a preceding prepare.

Beyond these optimizations, BTP also introduces the notion of Qualifiers. A Qualifier is a way of providing additional extended information within the protocol. Qualifiers can not only be of standard types such as *timeouts* for how long a participant is willing to remain in a prepared state, but can also be extended to provide new implementations that are better suited to different applications or participants.

## 2.2 Web Services Coordination and Transactions protocols (WS-C/T)

Another specification that aims at Web Services Transaction processing is the Web Services Coordination (WS-C) and Web Services Transactions (WS-T) protocols provided by IBM, Microsoft and BEA in 2003.

WS-C defines a generic coordination framework that can support arbitrary coordination protocols. It is extensible at both the coordinator level and the context level. Context information is specific to the type of coordination being performed. For example, while the basic WS-C context may be related to two-phase commit (2PC), a coordinator that executes a three-phase commit protocol can be easily plugged into a WS-C implementation with the context information enhanced if necessary. The WS-C framework exposes an Activation Service that supports the creation of coordinators for specific protocols and their associated contexts, a Registration Service that allows participants to register to receive protocol messages associated with a particular coordinator, and management of propagation of contextual information.

The WS-T specification plugs into WS-C and proposes two common industry completion patterns: Atomic Transactions (AT) that map to the existing transaction standards with traditional ACID semantics, and Business Activities (BA) that provide flexible transaction properties and are especially suitable for long-duration interactions where holding on to resources is impossible or impractical.

Similar to BTP, WS-T also provides several protocol optimizations. Atomic Transactions supports the read-only, one-phase and autonomous participant decision optimizations, while the Business Activities supports flexible consensus groups and participant resignation. The counterpart of BTP *qualifiers* in WS-T is handled by WS-Policy, where a *policy* is a standardized mechanism to advertise the operational characteristics.

## 3 Towards Adaptable Web Transaction Processing

Both of current proposed Web Services transaction specifications BTP and WS-C/T incorporate some kinds of optimizations to their basic two-phase completion protocols, and provide mechanisms such as *qualifiers* or *policies* for additional transaction

properties and qualifications of the protocols. However, these treatments are rather ad hoc and limited approaches. In order to achieve the goal of interoperability between heterogeneous TP systems and models, we advocate a distributed transaction processing architecture that is self-adaptive to application environment in a principle manner by means of extending the CORBA Activity Service Framework.

The OMG's CORBA Activity Service Framework represents the state-of-the-art in distributed object transactions. The service provides a framework for the construction of new middleware services used to support specific extended Unit of Work (UOW) models. The design of the service is based on the insight that the various extended transaction models can be supported by providing a general purpose event signaling mechanism that can be programmed to enable *activities* – application specific units of computations – to coordinate each other in a manner prescribed by the transaction model under consideration. The Framework has the ability to support a wide variety of extended transaction models, in that different models can be mapped onto specific implementations within this framework. Within the Activity Service Framework, a computation is viewed as composed of one or more activities that may be transactional, or may use weaker forms of serializability, or even not be transactional at all. Associated with each activity is an *activity coordinator* that can coordinate the execution of constituent activities. One or more *signal sets* are associated with a coordinator, and each signal set implements a specific coordination protocol, such as the two-phase commit (2PC) protocol. *Signals* are activity specific data that can be used to infer a flow of control during the execution of an application. Constituent activities are required to register themselves with a given signal set of the coordinating activity, by means of registering *actions* with the signal set. At an appropriate time, the coordinating activity triggers the execution protocol implemented by one of its signal sets by invoking a standard operation. This leads to the set signaling each registered activity by invoking an operation on the registered action. The signaled activity can now perform some specific computation and return results, and thus the protocol advances.

The flexibility to support a wide variety of extended transaction models offered by the Activity Service is particularly desirable in the Web Services world, where transaction scope may extend beyond individual enterprises. However, the Activity Service model is based on a standard object communications, rather than message communications as required by Web Services. In addition, although it provides a low-level infrastructure capable of enabling construction of various forms of extended transaction models, the framework is not adaptable at the run time to the ever-changing requirement of applications. To address these problems, we present an *Adaptive Transaction Framework* that extends the Activity Service model in the following way:

1. Rather than based upon object communication model of CORBA, interactions within the *Adaptive Transaction Framework* will be based on XML and SOAP messages. The message communications are defined by WSDL files instead of CORBA IDL. According to the Activity Service model, a Web Service (one or more of its WSDL operations) to be used would need to be registered as an *action* with the *activity coordinator* of the transaction originating Web Service. The

- registration describes an interest to receive transaction *signals* of a defined signal set, and to communicate back *action* outcomes. Such registration for signals is a necessary contractual agreement between Web Services that want to participate in the same transaction;
2. Each *signal set* in the Activity Service model implements a *single* specific extended transaction model as a state machine. Thus it prevents the framework from achieving adaptability at run-time. We extend the notion with *extended signal set*, which can generate signals of more than one extended transaction models. Thus it allows different transactional behaviors coexist in one instance of Web Services transaction. In addition, it makes possible that the transaction strategies adopted by certain participants of a Web Services transaction can be changed at the run time, either providing a solution to the heterogeneity problem or improving the transaction's Quality of Service;
  3. Use of Web Services Policy (WS-Policy) as a means to store the meta-information of the Web Services transactions. WS-Policy is a general framework that can be used and extended to describe a broad range of Web Services *policies*, each of which is a collection of one or more policy assertions. In our *Adaptive Transaction Framework*, we make use of the WS-Policy basically under two circumstances:
    - Each participant of the Web Services transaction provides its *participant policy* to indicate its individual preference and capability to support different transaction models and behaviors.
    - Each *extended signal set* has a *signal set policy* associated with it. This policy is provided by the *activity coordinator* to describe when and how the transaction strategies contained in the extended signal set are changed;
  4. Similar to the *signal set* in the Activity Service, an *extended signal set* is also assumed to implement a state machine, except that there are some *switch points* where the execution can consult the *signal set policy* or *participant policy* to determine whether or not it is possible and preferable to change the current transaction strategy to a new one. If so, the next signal generated by the extended signal set would be in accordance with the new strategy. However, there may be danger due to system failures during the process of transaction strategy changes. In order to guarantee correctness and integrity, a *strategy change transaction* may be needed to fulfill the switch work. Only when the *strategy change transaction* successfully commits will the new strategy be adopted, otherwise the TP system will be rolled back to the original strategy gracefully.

The following two examples illustrate in more detail on how the *Adaptive Transaction Framework* will work in the Web Service transaction domain:

**Example 1:** *Dynamic Presumption (DPr) in traditional ACID transactions*

Two-phase commit (2PC) is the de facto standard atomic commit protocol in traditional

ACID transactions. There are various optimizations to the basic 2PC, such as Presumed Commit (PrC) and Presumed Abort (PrA). The DPr protocol presented in [13] allows participants with different 2PC optimization presumptions to be dynamically combined in one Web Service transaction. It determines the presumptions at runtime, just before 2PC starts.

To cater for DPr in our *Adaptive Transaction Framework*, a `DPrExtendedSignalSet` should be constructed. It is similar to the `2PCSignalSet` described in [14], except that

- In addition to the signals defined in the `2PCSignalSet`, different “forced log signals” [13] should be included into the signal set, such as the `ForcedInit`, `ForcedPrepare`, `ForcedCommit` signals. This guarantees that `DPrExtendedSignalSet` supports both standard 2PC (with Presumed Abort optimization) and DPr (support dynamic combination of Presumed Abort and Presumed Commit optimizations);
- A *switch point*, namely, a “presumption switch point” is inserted in the state machine just before the “prepare” signal is sent to the participants. This switch point will force the *activity coordinator* to check the registered Web Services of their *participant policies* before sending the “prepare” signal, and thus determine the presumption adopted by individual Web Services.

Suppose there are two Web Services that take part in a DPr transaction. Web Service 1 takes the following *participant policy*:

```
<wsp:Policy xmlns:wsp="..." xmlns:txpolicy="...">
  <wsp:ExactlyOne wsp:Usage="Required">
    <txpolicy:2PCOptimization wsp:Preference="10">
      <txpolicy:2PCType>txpolicy:PresumedAbort</txpolicy:2PCType>
    </txpolicy:2PCOptimization>
    <txpolicy:2PCOptimization wsp:Preference="100">
      <txpolicy:2PCType>txpolicy:PresumedCommit</txpolicy:2PCType>
    </txpolicy:2PCOptimization>
  </wsp:ExactlyOne>
</wsp:Policy>
```

It shows that Web Service 1 supports both PrC and PrA optimizations, but with different preferences. It prefers PrC to PrA, may be due to the fact that disk write is its performance bottleneck, and PrC has less forced disk writes in the commit case. This means that Web Service 1 will choose to use the PrC strategy if the *extended signal set* it registers with supports PrC optimization, or else PrA strategy will be applied instead.

Similarly, Web Service 2 provides its *participant policy* as:

```
<wsp:Policy xmlns:wsp="..." xmlns:txpolicy="...">
```

```

<txpolicy:2PCOptimization wsp: Usage="Required">
    <txpolicy:2PCType>txpolicy:PresumedAbort</txpolicy:2PCType>
</txpolicy:2PCOptimization>
</wsp:Policy>

```

This participant policy simply requires the usage of PrA optimization.

Figure 1 and Figure 2 shows the message exchanges involved when the DPr transaction commits successfully in this example.

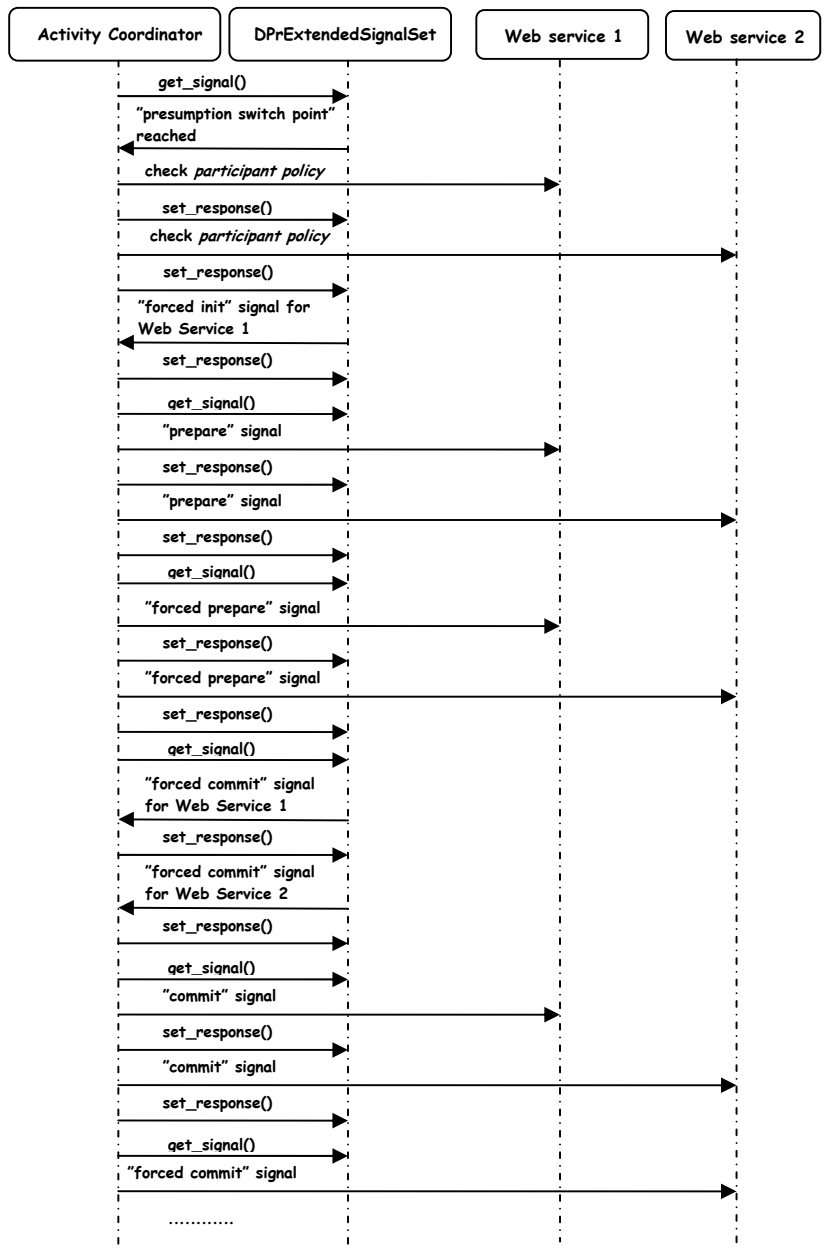


Figure 1. Message Exchanges in Example 1 (DPr)

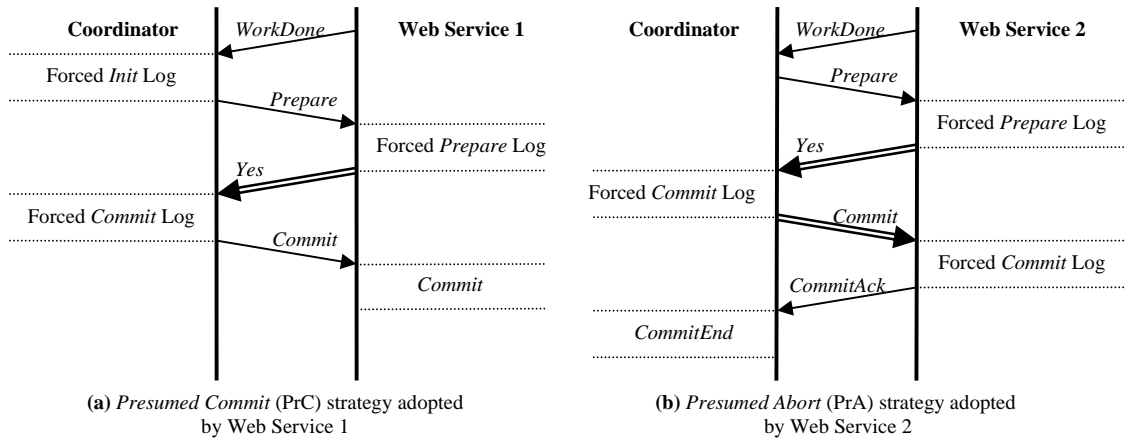


Figure 2. PrC and PrA strategies in Example 1

The above two figures illustrate message exchanges between different parties of DPr protocol in example 1. As Figure 1 shows, the activity coordinator initiates the commit by invoking *get\_signal* operation of its DPrExtendedSignalSet. The set, before sending *prepare* signal, finds out that a *presumption switch point* is reached in the state machine. It then notifies the activity coordinator, who checks its *signal set policy*, and finds out that DPr protocol should be applied. After consulting the corresponding *participant policies*, DPrExtendedSignalSet decides that PrC strategy should be applied to Web Service 1, while PrA be applied to Web Service 2. Thus the signal set starts to send signals to these two participants according to DPr protocol (Figure 2). First a *forced init log* signal is sent to the coordinator for necessary initial logging of Web Service 1 as the PrC strategy requires. When the response – done, rather than error in this case – is communicated to the set, the DPrExtendedSignalSet sends the next signal, *prepare*, to both registered Web Services. This process advances until the protocol ends.

### Example 2: One-phase optimization in non-ACID extended transactions

If an *activity coordinator* originally registered for a two-phase completion protocol, but somehow finds out that it has only one single participant Web Service, then it can tell the participant to confirm without a preceding *prepare*.

The following example shows one possible scenario: Two Web Services originally participate in a “normal” two-phase completion protocol, but because of its read-only nature, one of the participants resigns the transaction before preparation phase begins. At this point, there is only one participant remains and thus one-phase model should be applied.

This can be incorporated in the *Adaptive Transaction Framework* by a OnePhaseExtendedSignalSet which is similar to the 2PCSignalSet described in [14], except that a “one-phase switch point” is located just before the *prepare* signal is sent. This switch point makes the *activity coordinator* to check the possibility of one-phase optimization.

## 4 Related Work and Discussions

In the domain of object transaction models, some meta transaction models have been proposed, trying to describe and implement multiple transaction models.

Among these, ACTA [15] is a formalized framework intended to specify, analyze and synthesize extended transaction models. ACTA allows for specifying the structure and behavior of transactions as well as reasoning about the concurrency and recovery properties of the transactions. It uses a small set of modeling primitives to specify various extended transaction models. The five building blocks in ACTA used to specify essential components are history, inter-transaction dependencies, transaction conflicts, transaction view and delegation. However, the description in ACTA is very formal so that it can hardly be used directly for implementation of different transaction behaviors.

ASSET [16] is a flexible transaction facility based on a set of transaction primitives inspired by ACTA. The set of primitives enables realization of different extended transaction models. Using ASSET, an application programmer can construct extended transactions from scratch by properly composing the available primitives. Yet interoperability and adaptability of various transaction behaviors are not its focus.

A reflective transaction framework presented by [17] as a practical and modular method to implement extended transaction models. Modularity is achieved by applying the Open Implementation approach to the design of reflective transaction framework, and it extends the commercial TP monitor (Transarc's Encina) by providing transaction adapters as add-on modules built on top of existing commercial TP components. However, dynamic change of transaction behavior is not supported. In addition, the framework does not address *distributed* transaction processing, and thus is difficult to be applied to the Web Services context.

In summary, there is very little work concerning co-existence and interoperability of different TP systems, which is a very important issue and becomes even more crucial with the advent of Web Services. Both of the emerging Web Services transaction specifications, BTP and WS-C/T, try to address the heterogeneity and interoperability issue, yet there is still a lot of work to do to achieve the goal.

## 5 Conclusion and Future Work

We propose an *Adaptive Transaction Framework* in the Web Services domain by extending the OMG CORBA Activity Service Framework from an object transaction model to a Web transaction model. It is aimed to provide a generalized, principled mechanism to achieve interoperability among various distributed transaction processing models, and make the TP system able to automatically tune itself and apply the most appropriate transaction models and optimizations, in order to maximize the utilization of its own local resource and the satisfaction of its combined set of executing cooperators.

In addition, compatible transaction behaviors are able to co-exist in the system, providing more flexibility to different participants of Web Services transactions.

We are now in the stage of defining the *Adaptive Transaction Framework* systematically, and evaluating it by applying the framework to various Web transaction scenarios. Prototype implementation will be based on the Web Services domain, and incorporated into middleware platform.

This work is part of the Arctic Beans project [19], whose primary aim is to provide a more open and flexible enterprise component technology, with intrinsic support for configurability, re-configurability and evolution.

## References

- [1] World Wide Web Consortium (W3C), *Simple Object Access Protocol (SOAP)*, <http://www.w3.org/2000/xp/Group/> .
- [2] World Wide Web Consortium (W3C), *Web Services Definition Language (WSDL)*, <http://www.w3.org/2002/ws/> .
- [3] OASIS (Organization for the Advancement of Structured Information Standards), *Universal Description, Discovery and Integration of Web Services(UDDI)*, <http://www.uddi.org> .
- [4] Weihai Yu, “Reflective Transaction Processing – Motivations and Issues”, *NIK2001, Norsk Informatikk Konferanse*, 2001.
- [5] A. K. Elmagarmid (ed), “Transaction models for advanced database applications”, *Morgan Kaufmann*, 1992.
- [6] H. Garcia-Molina and K. Salem, “Sagas”, in *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 1987.
- [7] S. K. Shrivastava and S. M. Wheeler, “Implementing fault-tolerant distributed applications using objects and multi-coloured actions”, in *Proceedings of 10th Intl. Conf. on Distributed Computing Systems, ICDCS-10*, Paris, June 1990.
- [8] G. Weikum, H.J. Schek, “Concepts and Applications of Multilevel Transactions and Open Nested Transactions”, in *Database Transaction Models for Advanced Applications*, ed. A.K. Elmagarmid, *Morgan Kaufmann*, 1992.
- [9] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Gunthor and C. Mohan, “Advanced transaction models in workflow contexts”, in *Proceedings of 12th Intl. Conf. on Data Engineering*, New Orleans, March 1996.

- [10] OASIS *Business Transaction Protocol (BTP)*, <http://www.oasis-open.org/committees/business-transactions/>
- [11] IBM, BEA and Microsoft, *Web Services Coordination and Web Services Transaction specifications (WS-C/T)*, <http://msdn.microsoft.com/webservices/understanding/specs/default.aspx>
- [12] OMG, “Additional Structuring Mechanisms for the OTS ” (CORBA Activity Service), <http://www.omg.org/docs/orbos/01-11-08.pdf>
- [13] Weihai Yu, Yan Wang, Calton Pu, “A Dynamic Two-Phase Commit Protocol for Self-Adapting Services”, *2004 IEEE International Conference on Services Computing (SCC 2004)*, Shanghai, September, 2004
- [14] I. Houston, M. C. Little, I. Robinson, S. K. Shrivastava, S. M. Wheeler, “The CORBA Activity Service Framework for Supporting Extended Transactions”, *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Heidelberg, Germany, 2001
- [15] Chrysanthis, P. K., and K. Ramamritham, “ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior”, in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, May 1990.
- [16] Biliris, A., Dar, S., Gehani, N. H., Jagadish, H. V., Ramamritham, K., “ASSET: A System for Supporting Extended Transactions”, in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1994.
- [17] Barga, R. and C. Pu, “A Practical and Modular Method to Implement Extended Transaction Models”, in *Proceedings of 21<sup>st</sup> VLDB*, 1995.
- [18] IONA Technologies, introduction to Web Services, [http://www.iona.com/products/web\\_services\\_technology.htm](http://www.iona.com/products/web_services_technology.htm)
- [19] Andersen, A., Blair, G., Goebel, V., Karlsen, R., Stabell-Kulø, T., Yu, W., Arctic Beans, Configurable and Re-configurable Enterprise Component Architectures, in *IFIP/ACM International Conference on Distributed Systems Platforms, Middleware 2001*, Heidelberg, Germany. Also IEEE Distributed Systems Online, Volume 2, No. 7, 2001, <http://dsonline.computer.org>
- [20] C. Pu, et al. “Split-transactions for open-ended activities”, in *Proceedings of the Fourteenth International Conference on Very Large Data Bases*, Los Angeles, August 1988.