A Historic Survey of Transaction Management From Flat to Grid Transactions

Ting Wang and Paul Grefen

Subdepartment of Information Systems
Department of Technology Management
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{t.wang, p.w.p.j. grefen}@tm.tue.nl

Abstract

This is a working paper for the XTraConServe (XTC) Project, which aims to develop concepts and support facilities for a Business Transaction Framework (BTF) that utilizes abstract transactional constructs (ATCs) to provide a generic foundation for support of complex transactional services in contract-driven inter-organizational business interactions that rely on dynamically composed web services. In this paper, we investigate the classical transaction models and the key concepts and techniques with regard to transaction management. Some well-known work that has been or is being done in different application domains is reviewed following a time line, which reveals the development of transactions from the simplest no-structure model to the complex framework with hierarchical or layered structures.

Keywords: transaction; transaction model; transaction framework; transaction management

Table of Contents

1 Introduction

In this paper, we investigate the classical transaction models and the key concepts and techniques with regard to transaction management from a historic perspective. In this introduction, we first discuss the transaction concept. Next, we explain the basis for the timeline we use in our historic treatment. At the end of this introduction, we discuss the structure of this paper.

1.1 The transaction concept

What is a transaction? Actually, the concept of a transaction was invented as early as 6000 years ago, when Sumerians noted down and scribed on clay tablets in order to keep records of the changes of royal possessions. Generally speaking, a transaction is a transformation from one state to another. Over several thousand years, the concept has found its way into a broad range of disciplines. For example, in the business world, a transaction is defined as an agreement between a buyer and a seller to exchange an asset for payment. While in the database world, the real state of outside world is abstracted from and modeled by a database where the transformation of the state is reflected by an update of the database. From this perspective, a transaction can be defined as a group of operations executed to perform some specific functions by accessing and/or updating a database. These operations are in fact a kind of program designed to consistently interact with a database system. Later, with the wider use of transactional support in the IT domain, the original definition of a database transaction was generalized and extended by imposing a complex structure to support diversified applications.

In this paper, we use the term ‘transaction’ to refer to a reliable and coherent process unit interacting with one or more systems, independently of other transactions that provides a certain service or function for a running application. This new definition reflects the requirements for transactions that are able to capture more complex semantics arising from a broader range of application areas such as workflow management, Web services and Grid computing.

1.2 An overview of the history of transaction management

In this paper, we provide a survey of transaction management from a temporal perspective, i.e. we follow the history of transaction management from the ‘early dark days’ to current state of the art. In doing so, we distinguish between the following ‘ages’ in transaction management:

Stone Age. In the stone age, no explicit transaction management models and mechanisms were available. Reliability of business processes running on (database) systems was often not yet considered an issue at all. And if it was, its support was entirely the responsibility of application logic. As this age is not too interesting from a transaction management point of view, we do not pay attention to it in this paper.

Classic History. During the classic history, people realized that reliability of processes in multi-user, concurrent environments is an issue that deserves explicit attention – or rather *requires* explicit attention in order to keep things running correctly. In this age, the basic transaction model and mechanisms saw the light.

Middle Ages. In the middle ages, business application grew more complex and hence the requirements to transaction management rapidly increased. Consequently, the simple models and mechanisms developed in the classic history were not sufficient anymore. Consequently, a large variation of advanced transaction models and mechanisms supporting these requirements were developed for various application domains.

Modern Times. In modern times, we see the emergence of new application domains, in which the Internet usually plays a prominent role. To allow the proper operation of business processes in this new environment, transaction management has to be 'ported' to the Internet as well. This means, that the results of the previous ages of transaction management history are made fit for application in the Internet environment.

1.3 Structure of this paper

This paper gives an overview from the classical transaction models to the state-of-the-art work in the field of transaction management, moving from classic history to modern times.

We start the discussion of the classic history of transaction management in Section 2. We discuss the most basic transactions first used in a single database system – referred to as ACID transactions.

From Section 3, we proceed to the middle ages. In Section 3, we discuss the development of various extended transactions models, often referred to as advanced transaction models. In section 4, a synthesized meta-model for advanced transaction models, named ACTA, is described. Section 5 investigates a specific application domain: the transactional support for workflow systems. We apply a focus on the models proposed in the WIDE and CrossFlow projects, which closely relate to the XTC project.

From Section 6, we enter the modern times. Section 6 gives an overview of transaction support for the Web services world in the Internet environment. Afterwards, the work that is being done in Grid transactions is discussed in Section 7.

We end this paper with conclusions in Section 8.

2 ACID Transactions

As a database is an abstract representation that models part of a real organization and keeps its state consistent with the state of that organization, the programs interacting with the database need to reflect the requirements of real-world business. These requirements impose additional restrictions when designing a transaction. From the mid 1970s, some papers were published with early attempts to introduce these restrictions. This was the groundwork for the later defined properties generally known by the famous acronym 'ACID'. The ACID properties are the basis for the classic form of transactions known as 'traditional transactions' or 'flat transactions'.

2.1 ACID and VCRP properties

The ACID properties are [Haerder and Reuter 1983]:

Atomicity: A transaction either runs completely or has no effect at all, which means from the outside, that a transaction appears to have no observable intermediate states or it has never left the initial state.

Consistency: A transaction is a correct program and preserves all the integrity constraints. After the execution, the new state of the database complies with all the consistency constraints.

Isolation: A transaction is executed as if there are no other concurrent transactions. The effect of the concurrent transactions is the same as the effect when the transactions are executed serially.

Durability: A transaction completes successfully and thus makes a permanent change to the state of the database. Consequently, the results from a transaction must be able to be reestablished after any possible failures.

In fact, there is a more general corresponding representation of ACID properties: VCRP (Visibility; Consistency; Recovery; Permanence), which can be used as four measurements of transactions. Visibility represents the ability of an executing transaction to see the results of other transactions. Consistency refers to the correctness of the state of the database after a transaction is committed. Recovery means the ability to recover the database to the previous correct state in case of failures. Permanence is the ability of a successfully committed transaction to change the state of the database without the loss of the results when encountering failures. In [Warne 1993], the authors use these four notions to analyze and compare some transaction models such as nested transactions, sagas etc. This paper provides a standard framework to evaluate transactions by capturing the key characteristics of them.

2.2 Flat transactions

When we apply VCRP to evaluate the **traditional transactions** or **flat transactions** with no internal structures, we get the strict ACID properties that are essential for these relatively simple transactions. The underlying transaction processing (TP) system is responsible for ensuring the ACID properties. A TP system generally consists of a TP Monitor, which is a system program providing a middleware solution to manage transactions and control their access to a Database Management System (DBMS), one or more DBMSs and a set of application programs containing transactions [Lewis et al. 2002]. Atomicity and durability are guaranteed by the mechanism of recovery that is usually implemented by maintaining a log of update operations so that ‘redo’ and ‘undo’ actions can be performed when required. Isolation is guaranteed by the mechanism of concurrency control, which is implemented by using locks during the transaction process. A detailed overview of concurrency control and recovery techniques is available in [Ramamritham and Chrysanthis 1997]. Consistency is guaranteed by the integrity control mechanism usually provided by the TP system, though not complete in a strict sense.[1]

Flat transactions have proven to be very useful in traditional database applications where the execution time is relatively short, the number of concurrent transactions is relatively small and the database system only resides in one server. However, they lack the flexibility to meet the requirements of the applications developed later, for example, multi-database operations that need a certain level of transparency for the interactions with each local database or a workflow system that needs to support long-living transactions.

[1] There are two approaches to guarantee consistency. One implementation is to incorporate integrity control into DBMSs [Grefen 1993]. Another is to comply with the integrity constraints through the effort of application designers instead of TP systems [Gray and Reuter 1993].

3 Advanced transaction models

As mentioned in the previous section, ACID transactions, though very simple and secure, lack the ability to support the cases requiring long-living and/or complex transactions. Therefore a lot of advanced transaction models appeared to address such needs.

The basic idea of advanced transaction models is to divide a transaction into sub-transactions according to the semantics of the applications. These sub-transactions, also referred to as component transactions, can also be divided, if necessary, until every sub-transaction has a flat structure. The advanced transactions can perform more complex and longer-lasting tasks. For instance, when a failure occurs during a long-living transactional process, the system might restart from the middle of the transaction instead of the very beginning.

3.1 The save point concept

The partial rollback is supported by the mechanism of the **save point**, a concept first introduced in [Astrahan et al. 1976]. The authors suggested that during the execution of a transaction, a save point can be marked to return a save point number for subsequent reference. At each save point, special entries are stored containing the state of the database context in use by the transaction, and the identity of the lock acquired most recently. When a transaction fails, it can recover back to the recorded save point, where it restores the corresponding context and releases locks acquired after this save point. This way, rollback can return the system to a previous state in case of failure. There are some observations on the rollback mechanism using save points. For example, despite of the rollback of the database to the previously recorded state, the transaction's local variables are not rolled back, which means the transaction should adopt another alternative execution path after the rollback. Furthermore, after a rollback to one save point, the subsequently created save points are lost. Although the idea of persistent save point had been proposed to overcome the deficiency, it is hard to implement this idea in reality. For example, the database content can be rolled back to the previous state, but the local programming language variables will be lost. Another notice is that rollback is different from abortion. When aborted, the transaction is rolled back to the state when it started and the execution doesn't continue anymore. In contrast, a transaction rolled back to a save point still continues execution until it completes.

Although the save point mechanism can be used in combination with flat transactions, it gives more hints to the later development of advanced transaction models that have been proposed since the mid 1980s, i.e. distributed transactions, nested transactions, chained transactions etc. These models are more or less application specific, each of them addressing the need for a given situation. For example, if an organization needs to integrate several database systems residing in different servers to perform more comprehensive tasks in a multi-database system (MDBS), a distributed transaction or sometimes referred to as a multi-database transaction is needed. When considering complex-structured applications, a nested transaction properly addresses the need. For a time-consuming application with long-lasting transaction processes, a chained transaction is suitable to handle the problem. The above mentioned models are the examples of applying the idea of save point in different cases. A chained transaction is a variation of save points while the nested transaction is a generalization of save points [Gray and Reuter 1993].

3.2 Distributed and nested transactions

Distributed transactions consist of sub-transactions that may access multiple local database systems. Consequently, in addition to meeting integrity constraints in local systems, there are global integrity constraints imposed by the MDBS. Other concerns like global atomicity and isolation are also addressed. The whole transaction should be aborted if any sub-transaction fails. In [Breitbart et al. 1992], a most popular model at that time, 'base transaction model' was introduced and possible extensions to this basic model were proposed. The model defines two types of transactions, local ones and global ones. Several approaches to realize transaction atomicity and database consistency were discussed. Their work provides an overview of the most recent work until then in the MDBS area and raises some open problems for future research. Distributed transactions use a bottom-up approach to divide transactions into sub-transactions from a geographical point of view.

The most influential work underlying distributed transactions is the **X/Open Distributed Transaction Processing (X/Open DTP)** model, a software architecture developed by X/Open, a consortium of vendors who are defining portability standards for the UNIX environment. It allows multiple application programs to share resources provided by multiple resource managers, e.g. databases, and allows their work to be coordinated into global transactions [X/Open Ltd. 1996]. The X/Open DTP model is a standard for Two Phase Commit (2-PC) protocol, a key technology ensuring agreed outcome between participants in a distributed transaction. In the X/Open DTP model, the transaction manager, which is a functional component managing global transactions and coordinating the decision to start, commit or roll back, ensures atomicity at a global level, while each resource manager is responsible for ACID properties of its own resources.

In contrast to the distributed transactions, **nested transactions** adopt a top-down method to decompose a complex transaction into sub-transactions or child transactions according to their functionalities. The concept was first proposed in [Moss 1981] as the first discussion about programming transactions in a structured way. As it claims, nested transactions overcome the shortcomings of single-level transactions, for example, by permitting parts of a transaction to fail without necessarily aborting the entire transaction. The idea is that a transaction is composed of sub-transactions in a hierarchical manner, which means a sub-transaction can be divided into further sub-transactions if necessary, but only the leaf-level sub-transactions really perform database operations while others function as coordinators. A child transaction can only start after its parent starts and a parent can only commit after all its children have been terminated. The commitment of a child transaction is conditional on the commitment of its parents. Each child is atomic, thus it can abort independently regardless of its parent and siblings. When it aborts, the parent will take an action, like trigging another sub-transaction as an alternative. The aborted sub-transaction results as if it had not executed. It in fact changed the state of the database, and thus, a sub-transaction is not always consistent. However, the whole nested transaction still keeps the database consistent. The mechanism of the model is very powerful and has a strong relationship with the concept of modularization in software engineering [Gray and Reuter 1993]. This idea gained a lot of attention -- later on, there appeared some models based on it.

Based on the mechanism of nested transactions, in [Weikum and Schek, 1992], **multilevel transactions** (also called layered transactions) and their generalization, **open nested transactions**, were proposed. The authors present the concept of multilevel transactions as a variation of nested transactions where all transaction trees have their levels corresponding to the layers of the underlying system architecture. Note that the leaf nodes are all at the bottom level, i.e. the depths of these leaves are the same. They introduce the concept of pre-commit, which allows for the early commitment of a sub-transaction before the root transaction actually commits, thereby making it impossible to roll back in a traditional way. When a parent transaction needs to roll back a sub-transaction, it uses a compensating sub-transaction to semantically undo the committed one instead of using a state-based undo. Note that there are three differences from the nested transactions [Lewis et al. 2002]. First, children are executed only sequentially, not concurrently. Second, all the leaf-level sub-transactions are at the same bottom level in the transaction tree. Third, the commitment of a sub-transaction is unconditional, thereby making the result visible to other concurrently executing sub-transactions at the same level. Based on this model, if the structure of the transaction tree is no longer restricted to layering, thus leaves in different levels are allowed, multilevel transactions then evolve to open nested transactions. The authors investigated how open nested transactions relax the ACID properties to achieve the ideal orthogonality so that each of ACID properties can be omitted without affecting the others, to some extent. Compared to the nested transactions that guarantee global level isolation, which means the intermediate results of committed sub-transactions in nested transactions are invisible to other concurrently executing ones, open nested transactions relax the isolation property in the global level to achieve a higher level of concurrency.

3.3 Chained transactions and sagas

Although the nested transaction and its extensions are more powerful than the classical flat transaction, they are only fit for some specific environments like federated databases but are not suitable for environments requiring long-lived transactions. In such cases, the idea of **chained transactions** by decomposing a long running transaction into small, sequentially-executing sub-transactions was adopted. According to [Gray and Reuter 1993], the idea originates from IBM’s Information Management System (IMS) and HP’s Allbase database products. This idea is a variation of the save point mechanism that a sub-transaction in the chain roughly corresponds to a save point interval. However, the essential difference is that each sub-transaction itself is atomic, while each interval between every two save points is only part of an atomic transaction. In the chain, a committed sub-transaction triggers the next upon commitment, one by one, until the whole chained transactions commit. When encountering a failure, the previously committed sub-transactions would have already made durable changes to the database so that only the results of the currently executing sub-transaction are lost. This way the rollback only returns the system to the beginning of the most recently-executing sub-transaction. Notably, from the application perspective, the atomicity and isolation properties are no longer guaranteed by the whole chain. For example, in the middle of execution, all the committed sub-transactions cannot be undone, which leads to a problem to abort the whole chain. Another case is that other concurrent transactions can see the intermediate results generated during the execution of the chain.

Based on the idea of chained transactions, **Sagas** were proposed with combination of a compensation mechanism to roll back. The saga model described in [Garcia-Molina and Salem 1987] is a classical transaction model used as a foundation of many later transaction frameworks. Sagas divide a long lasting transaction into sequentially executed sub-transactions and each sub-transaction, except the last one, has a corresponding compensating sub-transaction. All these sub-transactions are atomic with ACID properties. When any failure arises, the committed sub-transactions are undone by those compensating sub-transactions. Unlike the non-atomic chained transactions that cannot undo the committed sub-transactions in the case of an abort, sagas can use compensating sub-transactions to return the whole transaction back to the very beginning. Note that the recovered start state is not exactly the same as the original start state but only equivalent to it from an application point of view. In this sense, sagas in a whole still preserve application-dependent atomicity. Similar to chained transactions, a saga transaction may be interleaved with other current transactions, thus isolation is not guaranteed. Consequently, consistency in sagas is not realized by serializability. Some extensions of saga models are introduced in [Chrysanthis and Ramamritham 1992] with more recovery options.

3.4 Conclusion on advanced transaction models

The above advanced transaction models can be viewed as various extensions to flat transactions that release one or more ACID constraints to meet with specific requirements. Through their different structures and applying environments, we can observe that there are two strategies adopted to extend the simple ACID transactions. One is to modularize a complex transaction with hierarchies. By this means, a big transaction is divided into smaller components, which can in turn be decomposed. This strategy has been applied in various transactions including distributed transactions, nested transactions, multilevel transactions, and open nested transactions. With the modularization of a complex transaction, the structure is clearer from a semantic perspective. Another strategy is applied in chained transactions, sagas etc through decomposing a long-lasting transaction into shorter sub-transactions. By means of splitting the long processing time, each transaction can be divided into a sequential series of smaller components that are operated in a shorter time thus minimizing the work lost during a clash.

However, the avalanche of the advanced models does not mean that flat transactions have been replaced by these more powerful models. On the contrary, because of their simple structures and easily implemented ACID properties, flat transactions still dominate the database world.

4 ACTA model

Besides the proposals of the transaction models discussed in the previous section, a novel attempt was made in [Chrysanthis and Ramamritham 1990], which develops a comprehensive framework named ACTA by unifying existing models to capture the semantics and reason for the concurrency and recovery properties of complex transactions.

4.1 Original ACTA framework

In the ACTA framework the behavior of active components (transactions) and passive components (objects) represents the behavior of a transaction system. Interactions among transactions are expressed in terms of effects, i.e. effects of transactions on other transactions and effects of transactions on objects they access.

Two types of effects that transactions have on other transactions are specified as ‘commit-dependency’ and ‘abort-dependency’. Commit-dependency describes the relationship of one transaction T1 on another transaction T2 that T1 cannot commit until T2 either commits or aborts. Abort-dependency describes the relationship of T1 on T2, that if T2 aborts, then T1 should also abort.

The framework captures the effects of transactions on objects by two objects sets and the concept of delegation. Every transaction is associated with a few objects contained in ‘view set’ or ‘access set’. View set contains all the objects potentially accessible to the transaction while access set contains the objects that are already accessed by the transaction. Transactions make changes on the objects through three forms of delegation, i.e. ‘delegation of state’, ‘delegation of status’ and ‘limited delegation’. Delegation of state describes the ability of a delegator (delegating transaction) to move the objects from its access set to the delegatee’s (delegated transaction) access set. Delegation of status represents the ability of the delegator to undo the changes on the objects before those objects are moved to the access set of the delegatee. Limited delegation implies the ability to make the changes to the objects persistent in the view set before adding them to the access set of the target transaction. Through the delegation mechanism, the visibility of objects can be controlled.

When conjuncting with commit and abort dependencies, delegation can also specify the recovery properties of a transaction model. This way, via formalized expressions describing the dependencies, object sets and delegations, ACTA allows for the specification of the structure and behavior of transactions as well as reasoning their concurrency and recovery properties.

4.2 Later developments

From the above description, ACTA is a meta-model that can be used to flexibly develop new transaction models. This approach inspired the later ASSET model proposed in [Biliris et al., 1994], which uses primitives at a programming language level based on ACTA building blocks such as ‘history’, ‘delegation’, ‘dependency’, ‘conflict set’ etc. However, the demonstration of the power of the ACTA framework by 5 transaction models seems inadequate. Not all the types of the popular models are included, for example, the saga and multilevel transaction models. In addition, its complexity also makes it difficult to implement.

5 Workflow transactions

Based on the advanced transaction models discussed in Section 3, specific transaction models have been designed for the support of business processes, usually identified as workflow transaction models or transactional workflows[2]. Below, we first explain the concept of transactional workflows. Next, we describe two example approaches that are highly relevant to the XTC project.

5.1 The transactional workflow concept

The concept of **transactional workflow** was first introduced in [Sheth and Rusinkiewicz 1993] to clearly state the relevance of transactions to workflows. Since the mid 1990s, two developments took place in the area of workflow technologies. One is the development of the transaction model supporting workflows and the other is the development of languages for workflow specification. From a transactional point of view, workflows are generalized extended transactions with focus on the automation of the complex, long-lasting business processes in distributed and heterogeneous systems. A workflow process may involve database transactions or human activities, so the ACID properties would not be the major concern anymore. Similar to the decomposition mechanism of advanced transaction models, a workflow process can be modeled by decomposition into some sub-processes in a hierarchical or sequential way. From this perspective, a workflow process can be viewed as a complex transaction hierarchically or sequentially consisting of sub-transactions and/or non-transactional tasks. A lot of work has been done in this area to address the need for transaction support in a process-centric environment.

5.2 WIDE model

In [Grefen et al. 1997], a two-layer transaction model, known as the **WIDE transaction model**, was presented. The bottom layer consists of local transactions with a nested structure that conform to the ACID properties [Boertjes et al. 1998]. The upper layer is based on Sagas that roll back the completed sub-transactions using the compensation mechanism, thus relaxing the requirement of atomicity. The semantics of the upper layer have been formalized using simple set and graph theory [Grefen et al. 2001]. The local transaction layer was designed to model low-level, relatively short-living business processes, whilst the global transaction was designed to model high-level long-living business processes.

This flexible approach was adopted later in [Vonk and Grefen 2003] in order to develop a more comprehensive X-transaction model. Note that these two models address the needs in different contexts. The WIDE transactional model caters for intra-organizational workflow while the X-transaction model can deal with specific inter-organizational workflow.

[2] The relation of workflows and transactions can be of various nature, depending on the point of view [Grefen 2002]. This has resulted in diverse approaches [Grefen 1999].

5.3 X-transaction model

The **X-transaction** model is a three-level, compensation based transaction model for inter-organizational workflow management in the CrossFlow project, where a contracted service outsourcing paradigm was supposed [Vonk and Grefen 2003]. The three levels in this model are the outsourcing level, contract level and internal level, each with a different visibility to the consumer or the provider.

The model views an entire workflow process as a transaction. For intra-organizational processes, they can be divided into smaller I-steps that adhere to ACID properties. Each I-step has a compensating step in case of failure. Similar to this idea, a contract-level cross-organizational process is divided into X-steps, each of which corresponds to one or more I-steps. The model also introduced a concept of the safe point, which is similar to the save points in Sagas. With the components of I-steps, X-steps and compensating steps, the X-transactional model realizes a flexible intra- or cross- organizational rollback effect so as to support all the scenarios with all the combinations of rollback scopes and rollback modes. An architecture to support this model was also proposed.

There are three layers in the architecture as in the transaction model, where a dynamically created upper layer is built on the top of the static layer, which involves local Workflow Management Systems. Between them, an isolation layer exists to provide portability with respect to specific WFMSs.

6 Web services transactions

From the late 1990s until now, more and more attention has been placed on the area of transactions in the loosely-coupled Web services (WS) world. In addition to other accepted standards such as SOAP, WSDL, UDDI etc., a technique to guarantee the consistency and reliability of WS applications is needed. However, there is no such mature transaction mechanism that is widely accepted as a standard. Currently, there are three possible candidates, which we discuss below. Next, we compare the approaches.

6.1 BTP, WS-Tx and WS-CAF

The first candidate is the **Business Transaction Protocol** (**BTP**) [Ceponkus et al. 2002] developed by OASIS, which, as its name shows, is not exclusively designed for Web services but also for non-Web services applications. BTP is an eXtensible Markup Language (XML) based protocol for representing and seamlessly managing complex, multi-step business-to-business (B2B) transactions over the Internet.

The second candidate is the **Web Services Transactions (WS-Tx)** specification consisting of WS-Coordination (WS-C) specification, WS-AtomicTransaction (WS-AT) specification and WS-BusinessActivity (WS-BA) specification initiated by Microsoft, IBM and BEA [Cabrera et al. 2004a, Cabrera et al. 2004b, Cabrera et al. 2004c]. WS-Tx specifications define mechanisms for transactional interoperability between Web services domains and provide a means to compose transactional qualities of service into WS applications. Among the specifications, WS-C describes an extensible coordination framework to coordinate the distributed applications. WS-AT specifies the coordination type for ACID transactions and WS-BA specifies the type for long-running business transactions.

The third candidate, **WS Composite Application Framework** (**WS-CAF**) [Bunting et al. 2003a], is also under the umbrella of OASIS initiated by a consortium consisting of SUN, Oracle, Arijuna etc., with the purpose of developing an interoperable, easy to use and implement framework for composite WS applications. Similar with WS-Tx, it is also a series of specifications consisting of WS Context [Bunting et al. 2003b], WS Coordination Framework [Bunting et al. 2003c] and WS Transaction Management [Bunting et al. 2003d].

6.2 Comparing the approaches

In [Little and Freund 2003], a comparison between BTP and WS-Tx is made. This paper shows how these two specifications both attempt to address the problems of running transactions with Web services. With a clear list of pros and cons, the authors make a comparative analysis of the two competitors in a table. At the end, they conclude that the two specifications differ in some critical areas such as transaction interoperability. It also concludes that BTP lacks ‘the ability to use existing enterprise infrastructures and applications and for Web services transactions to operate as the glue between different corporate domains’. Considering the fact that large strongly-coupled corporate infrastructures exist behind those loosely-coupled Web services, the authors call for the attention of leveraging ACID transactions, which underlying the internal corporate infrastructures, instead of replacing them with new models to design WS transactions.

Another comparison of the above mentioned three specifications is presented in [Kratz 2004]. This technical report gives a detailed overview of the three specifications and highlights the differences between the three candidates. At the end, the author points out the need for one open standard to realize the interoperability both in Web services and business areas, possibly by integrating the existing ones within the WS-CAF framework.

In [Jin and Goschnick 2003], it is stated that BTP is the most appropriate candidate to be an Internet transaction standard. The authors present an Agent Based Transactional (ABT) model by applying the 'Shadowboard Agent Architecture' to the wrapping of the existing services as Web services. They claim that using agent technology for transaction management is the right choice in the Web services environment. The benefit of the ABT model, according to the authors, is that it can flexibly choose alternative participants to reduce the rollback and compensation chances. This is a novel attempt to combine Web services with agent technologies. However, it is still in a preliminary stage. The implementation is only based on a simple scenario, which can be realized by existing technologies.

7 Grid transactions

Like an electricity power grid that pools together distributed electric energy, Grid computing is a form of distributed computing that involves coordinating and sharing of computing resources across the web globally. Because of its vision to create a worldwide network of computers that act as if they are one, Grid computing makes the exclusive immense computing power previously only available to a few organizations now available to everyone. This new emerging technology has been gaining a lot of attention from its birth.

With more and more projects launched, among them the largest one, CoreGRID commenced on 1 September 2004, a lot of research is available within the areas of infrastructure and middleware. Much less effort is spent in the area of Grid transactions, however. Below, we describe two efforts currently ongoing in this area.

7.1 Ongoing work in Grid transactions

TM-RG (GGF Transaction Management Research Group), initiated in Europe, is working on Grid transactions with the goal of investigating how to apply transaction management (TM) techniques to Grid systems. It is stated in the charter [Steinbach et al. 2004] that ‘a common grid transaction service would contribute a useful building block for professional grid systems’. The group is trying to implement possible Grid transaction approaches that may develop on the basis of Web services transactions as discussed in Section 6. However, there is no research output yet from this group.

In Shanghai Jiang Tong University, a group is working on a new service-oriented Grid Transaction Processing architecture called **GridTP** based on the Open Grid Services Architecture (OGSA) platform and the X/Open DTP model [Qi et al. 2004]. It is claimed that GridTP provides a consistent and effective way of making existing autonomously managed databases available within Grid environments.

One recently available paper [Türker et al. 2005] tackles some of the highlighting questions in this emerging research area in terms of how transactions fit into a grid environment. A protocol ensuring correct executions of concurrent applications in the global level with the absence of a global coordinator is proposed to realize the concept of distributed, peer-to-peer grid transactions. The approach in this paper is based on some known concepts and techniques, such as the recoverability criterion, serialization graph testing and partial rollback. The main idea of the approach is that dependencies between transactions are managed by the transactions, so globally correct executions can be achieved even without the complete knowledge gained from communications among dependent transactions and the peers they have accessed. The idea is innovative in the sense that it uniquely combines old concepts and techniques for a new purpose.

7.2 Comparing the approaches

It is hard to properly evaluate and compare the above approaches at this early stage in such a new area. We can expect that with the developments of Grid technology, the need for a standard protocol to provide transactional support will be on the agenda as Grid computing does need a reliable way to coordinate and communicate.

8 Conclusion

From the discussion of transaction management in this paper, a clear historical thread from the classic age to modern times is revealed reflecting the transition from database transactions to workflow transactions to grid transactions.

We notice that, with the development of information technology toward a broader geographical scope and larger scale, the future trend of transaction management is correspondingly following a direction to address the need for more functionalities and better performance in a distributed, heterogeneous, cross-organizational environment. This need is essentially prominent in an era witnessing a rapidly increasing e-business, which often involves multiple organizations all across the world dynamically establishing business relationships over the Internet.

However, despite the complex requirements developed through all these years, the fundamental idea that a transaction provides a reliable approach to achieve mutually agreed goals remains the same when designing new transaction models or frameworks.

References

Astrahan, M., et al., (1976). System R: A relational approach to database management. *ACM Transactions on Database Systems* 1(2): 97-137.

Biliris. A., et al., (1994). ASSET: A System for Supporting Extended Transactions. *Proceedings of ACM SIGMOD Conference on Management of Data*: 44-54, Minneapolis, MN.

Boertjes, E., et al., (1998). An Architecture for Nested Transaction Support on Standard Database Systems. *Proceedings 9th International Conference on Database and Expert System Applications (DEXA)*: 448-459.

Breitbart, Y., et al., (1992). Overview of multidatabase transaction management. *VLDB Journal* 1(2): 181-240.

Bunting, D., et al., (2003a). Web Service Context (WS-CTX).

Bunting, D., et al., (2003b). Web Service Coordination Framework (WS-CF).

Bunting, D., et al., (2003c). Web Services Composite Application Framework (WS-CAF).

Bunting, D., et al., (2003d). Web Services Transaction Management (WS-TXM).

Cabrera, L.F., et al., (2004a). Web Service Coordination (WS-Coordination).

Cabrera, L.F., et al., (2004b). Web Services Atomic Transaction (WS-Atomic Transaction).

Cabrera, L.F., et al., (2004c). Web Services Business Activity Framework (WS-Business Activity).

Ceponkus, A., et al., (2002). Business transaction protocol version 1.0.

Chrysanthis, P. K., and Ramamritham, K., (1990). ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior. *Proceedings of the ACM SIGMOD International Conference on Management of Data*: 194-203.

Chrysanthis, P. K., and Ramamritham, K., (1992). ACTA: The SAGA continues. In Elmagarmid, A., editor, *Database Transaction Models for Advanced Applications*. Morgan Kaufmann Publishers.

Garcia-Molina, H., and Salem, K., (1987). Sagas. *Proceedings of the ACM SIGMOD International Conference on Management of Data*: 249-259, San Francisco.

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

Grefen, P., and Apers, P., (1993). Integrity Control in Relational Database Systems - An Overview. *Journal of Data & Knowledge Engineering* (10)2: 187-223.

Grefen, P., et al., (1997). Two-Layer Transaction Management for Workflow Management Applications. *DEXA 1997*: 430-439.

Grefen, P., (1999). Advanced Architectures for Transactional Workflows - or - Advanced Transactions in Workflow Architectures. *Proceedings International Process Technology Workshop (IPTW)*.

Grefen, P., et al., (2001). Global Transaction Support for Workflow Management Systems: from Formal Specification to Practical Implementation. *VLDB Journal* (10)4: 316-333.

Grefen, P., (2002). Transactional Workflows or Workflow Transactions?, *Proceedings 13th International Conference on Database and Expert Systems Applications (DEXA)*: 60-69.

Haerder, T., and Reuter, A., (1983). Principles of transaction-oriented database recovery. *ACM computing Surveys* 15(4): 287-317.

Jin, T., and Goschnick, S., (2003). Utilizing Web Services in an Agent-based Transaction Model (ABT). *International workshop on Web Services and Agent-based Engineering (WSABE-2003), held in conjunction with AAMAS-2003*: 1-9, Melbourne, Australia.

Kratz, B., (2004). Protocols For Long Running Business Transactions.

Lewis, P. M., et al., (2002). *Databases and Transaction Processing: An Application-Oriented Approach*. Addison-Wesley, United States

Moss, J. E. B., (1981). *Nested Transactions: An Approach to Reliable Distributed Computing*. PhD thesis, EECS Department, M.I.T.

Qi, Z., et al., (2004). Integrating X/Open DTP into Grid Services for Grid Transaction Processing. *10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'04)*: 128-134, Suzhou, China.

Ramamritham, K., and Chrysanthis, P. K., (1997). *Advances in Concurrency Control and Transaction Processing*. IEEE Computer Society Press, Los Alamitos, California.

Sheth, A., and Rusinkiewicz, M., (1993). On Transactional Workflows. In Hsu, M., editor (1993). *Special Issue on Workflow and Extended Transaction Systems*, volume 16. IEEE Computer Society, Washington, DC.

Steinbach T., et al., (2004) *Proposed Grid Transactions RG – Charter* at http://www.data-grid.org/tm-rg-charter.html

Türker, C., el al., (2005). How can we support Grid Transactions? Towards Peer-to-Peer Transaction Processing. *Proceedings of the 2005 CIDR Conference*, Asilomar, California.

Vonk, J., and Grefen, P., (2003). Cross-Organizational Transaction Support for E-Services in Virtual Enterprises. *Distributed and Parallel Databases* 14(2): 137-172.

Warne, J., (1993). *An extensible transaction framework: Technical overview*. Cambridge, U.K.

Weikum, G., (1991). Principles and realization strategies of multilevel transaction management. *ACM Transactions on Database Systems* 16(1): 132-180.

Weikum, G., and Schek, H., (1992). Concepts and Applications of Multilevel Transactions and Open Nested Transactions. In Elmagarmid, A., editor, *Database Transaction Models for Advanced Applications*. Morgan Kaufmann Publishers.

X/Open Company Ltd., (1996). *Distributed Transaction Processing: Reference Model, version 3*. X/Open Company Ltd., U.K.