Distrib Parallel Databases (2008) 23: 235–270
DOI 10.1007/s10619-008-7028-1

A survey on the history of transaction management: from flat to grid transactions

Ting Wang · Jochem Vonk · Benedikt Kratz · Paul Grefen

Published online: 24 April 2008

Abstract Transactions have been around since the Seventies to provide reliable information processing in automated information systems. Originally developed for simple 'debit-credit' style database operations in centralized systems, they have moved into much more complex application domains including aspects like distribution, process-orientation and loose coupling. The amount of published research work on transactions is huge and a number of overview papers and books already exist. A concise historic analysis providing an overview of the various phases of development of transaction models and mechanisms in the context of growing complexity of application domains is still missing, however. To fill this gap, this paper presents a historic overview of transaction models organized in several 'transaction management eras', thereby investigating numerous transaction models ranging from the classical flat transactions, via advanced and workflow transactions to the Web Services and Grid transaction models. The key concepts and techniques with respect to transaction management are investigated. Placing well-known research efforts in historical perspective reveals specific trends and developments in the area of transac-

Recommended by Ahmed K. Elmagarmid.

T. Wang · J. Vonk (✉) · P. Grefen
Subdepartment of Information Systems, Department of Technology Management, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
e-mail: j.vonk@tm.tue.nl

T. Wang
e-mail: t.wang@tm.tue.nl

P. Grefen
e-mail: p.w.p.j.grefen@tm.tue.nl

B. Kratz
Infolab, Department of Information Systems and Management, Tilburg University, P.O. Box 90153, 5000 LE Tilburg, The Netherlands
e-mail: B.Kratz@uvt.nl

tion management. As such, this paper provides a comprehensive, structured overview of developments in the area.

Keywords Transaction management · Transaction model · Workflow transaction · Web services transaction · Grid transaction · Historic survey

1 Introduction

Transactions have been generally around since the Seventies to provide reliable information processing in automated information systems. No real-world, multi-user administrative information system can be thought to exist without a proper transaction management subsystem that prevents the system from entering undesired states as an effect of system failures, user errors or interference of concurrently executing actions. Transaction mechanisms were originally developed to support simple 'debit-credit' style database operations in centralized systems, often running applications of a financial nature. From there, transactions have moved through the decades into much more complex application domains including aspects like distribution, process-orientation and loose coupling. Nowadays, transaction mechanisms are developed that should support complex, interorganizational business processes that are executed by dynamically composed business networks.

The amount of published research work on transactions is huge and a number of overview papers and books already exist. Especially in the Nineties, some well-known overview books were produced [25, 30, 43]. A concise but complete historic analysis providing an overview of the various phases of development of transaction models and mechanisms in the context of growing complexity of application domains is still missing, however. The mentioned overview books, for example, were all written before the 'Internet age' and therefore miss the development of Internet-based transaction mechanisms. More recent work on Internet transactions is usually not placed in enough historic perspective. To fill this gap, this paper presents a historic overview of transaction models organized in several 'transaction management eras'.

This paper investigates numerous transaction models ranging from the classical flat transactions, via advanced and workflow transactions to the Web Services and Grid transaction models. The key concepts and techniques with respect to transaction management are investigated. This paper does, however, not focus on technical details of each transaction model, protocol or framework (this would be infeasible in one single paper). Rather, it stresses main ingredients and trends allowing to describe important developments. Placing well-known research efforts in historical perspective reveals specific trends and developments in the area of transaction management. The aim is not to be complete in describing all efforts (again, this would not be feasible), but to bring an up-to-date picture and present a clear thread by analyzing and comparing the works in the domain of transaction management. Within each era, an illustrative example is given for a selected transaction model to clarify main aspects of transaction management development representative of that era.

The goal of this paper is not to introduce new theories, models, or mechanisms. Neither is its aim to redo the overview works that have appeared in the past. This

paper does present a concise historical analysis of the field of transaction management ranging from its earliest days to the latest current developments. As such, it provides an overview that allows to understand various developments and easily interrelate them.

1.1 Structure of this paper

This paper is structured as follows. First, the transaction concept is introduced in Sect. 2. In this section, we also introduce the eras in which we have classified the transaction models: the Stone Age, Classical History, Middle Ages, Renaissance and Modern Times. The Stone Age is the age without transactions. Sections 3 to 7 present the main classes of the transaction models, coupled to the other four eras. The first three of these sections coincide with the first three eras. To the Modern Times (covering Web developments), we have devoted two sections. As such, Sects. 3 to 7 cover respectively the classical flat transaction model, advanced transaction models, workflow transactions, Web Services transactions, and Grid transactions. As transaction frameworks can be applied in a broader sense than transaction models, they cannot be placed in the main classification of the eras. Therefore, they are discussed separately in Sect. 8. The paper ends with conclusions and a brief look into the future.

2 Concept and history of transactions

Before transaction models can be classified into eras, this section first presents the transaction concept and its early historic background, after which the separate ages are introduced together with the reasoning why we have come to those ages.

2.1 The transaction concept

What exactly is a transaction? 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 and trades [69]. 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 the 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 extended and generalized by using complex structures to support a wide range of 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 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.

In contrast to transaction support for corporate information systems, mobile transactions deal with other aspects concerning transaction support, for example disconnected devices (considered as normal situations, instead of exceptional situations), movement of transactions, and bandwidth variations. An elaborate survey on mobile transactions can be found in [65]. As mobile devices have only recently become more commonplace, mobile transactions are a recent development, much like the web and grid transactions. It is therefore natural that the developments in the first three eras of transaction management support serve as a basis for mobile transactions. For example, the Moflex transaction model [44] expanded the flexible transaction model [24] and Kangaroo transactions [20] are build on global transactions in a multidatabase environment [12] and split transactions [61]. As we focus on corporate information systems, the area of mobile transactions is considered out of scope for this paper.

2.2 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 the 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 models and mechanisms were available. Reliability of applications 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 applications 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. Reliability of (stored) data and its manipulation was the main focus of the transaction support in this era. Definitions and common terminology related to the transaction model and mechanisms can be found in standard work from that era, for example [19, 32].
- *Middle ages*
  In the middle ages, business applications grew more complex and hence the requirements on transaction management rapidly increased. The simple model 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. The focus of transaction management changed from pure data towards functions. This means that the semantics or function of the application determined the correctness of executing state changes. Exceptions in application execution were handled by transaction management which brought the application back into a consistent state (while the data may not be). Explanations of common terminology from this

era (e.g., distributed systems, multi-database systems, homogeneous and heterogeneous distributed database systems) can be found in standard works covering that era, for [25, 66].

- *Renaissance*

  The renaissance saw the combination of process control with transaction support, leading to numerous transaction models specially suited to support process control systems, like workflow management systems (WfMS). Originating from two different research areas, i.e., process control and transaction support, these models are either referred to as transactional workflows or as workflow transactions [33]. In this era, the focus of transaction management support changed from functions to process. Reliability in process execution was the main issue. Definitions and descriptions of common terminology (e.g. process oriented systems, intra-, cross-, and inter-organizational processes, workflow management) from this era are presented in, for example [4, 43, 51].

- *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 from the previous transaction management eras are made fit for applications in the Internet environment. Although processes are still a key component in this era, the focus moved towards transaction support for collaborative, autonomous parties. A frame of reference for common terms, definitions and issues from this era, like services, Web services, loosely coupled services, is for example presented in [3, 60].

Figure 1 illustrates the transaction ages in relation to each other positioned on a timeline. The specific transaction models, mechanisms, and properties covered in the sequel of this paper are listed under the specific age to which they belong.

3 Classic history: the classical transaction model

At the time in which databases became more commonplace and evolved into large multi-user, concurrently accessible, repositories, people started realizing that a mechanism was necessary that automatically ensures and enforces the consistency of the database.

3.1 ACID properties

Already identified in the beginning of the 1980s, see e.g., [32, 38] are the ACID properties for transactions. They were considered fundamental, at that time, to provide for robust and reliable database operations in concurrent settings. The ACID properties are:

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

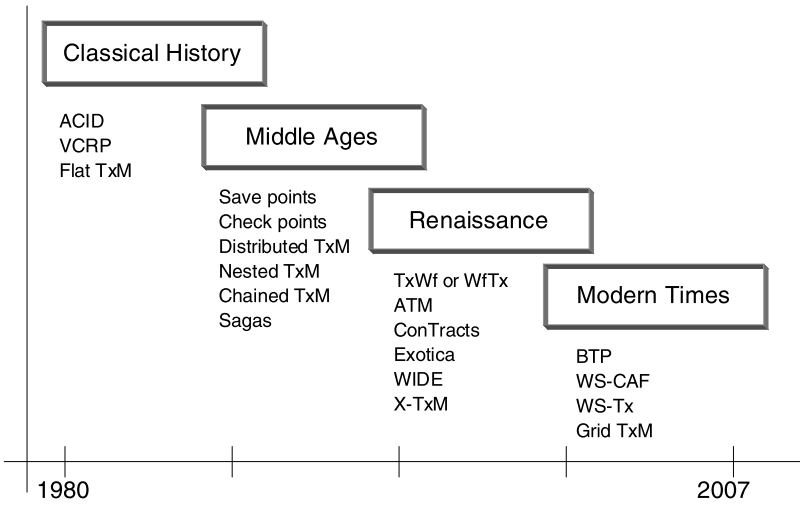

Fig. 1 Ages in transaction management

- 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.

Transactions adhering to the ACID properties are guaranteed to be failure atomic and serializable. This means that each transaction is guaranteed to execute in its entirety or not at all and that the outcome of operations performed within a transaction is the same as if these operations would be performed in a sequence (a series of executions, one after the other).

3.2 VCRP properties

In fact, VCRP (Visibility; Consistency; Recovery; Permanence) represent ACID properties in a more general way. 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 when failures occur. 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.

The VCRP properties can be used to evaluate transaction models. For example, in [77], the authors use these four notions to analyze and compare some transaction

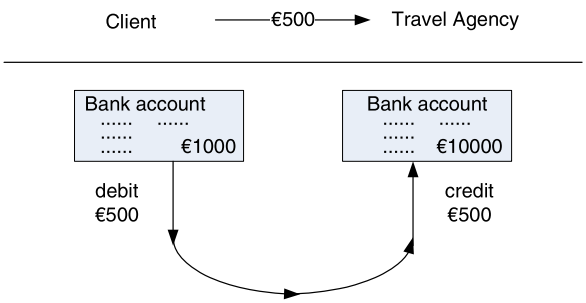

Fig. 2 Flat transaction model example

models such as nested transactions and sagas. By capturing the VCRP properties of the transactions, the author provides a standard framework to evaluate them.

3.3 Flat transactions

When we apply VCRP to evaluate traditional transactions, which are also known as flat transactions as they have no internal structures, we get the strict ACID properties that are essential for these relatively simple transactions. The transaction processing (TP) system is responsible for ensuring the ACID properties. A TP system generally consists of [49]:

- a TP Monitor, which is an application to manage transactions and control access to a Database Management System.
- one or more Database Management Systems (DBMS). However, for flat transactions only one DBMS can be part of the TP system.
- a set of application programs containing transactions.

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 [63]. Consistency is guaranteed by the integrity control mechanism usually provided by the TP system, though not complete in a strict sense.[1]

An illustrative example for the usability of flat transactions is the classic *Bank Transfer* scenario, see also Fig. 2. In this example, a client of a travel agency booked a holiday trip for a cost of €500, to be paid by bank transfer. What happens in this bank transfer is that the €500 are first debited from the client's bank account and then credited to the agency's bank account. Both operations should succeed together or none of the two should succeed, i.e., an atomic operation, resulting in a bank account balance of €500 for the client and €10500 for the travel agency (successful transfer) or €1000 for the client and €10000 for the travel agency (unsuccessful transfer).

[1] There are two approaches to guarantee consistency. One implementation is to incorporate integrity control into DBMSs [27]. Another is to comply with the integrity constraints through the effort of application designers instead of TP systems [30].

Consistency is guaranteed by making sure that the total amount of money from both accounts (€11000) are the same before and after the debit and credit operations.

During the operations, the intermediate results are 'hidden' to other concurrently running transactions (concurrency control), so that it is not possible to observe a situation in which the client account is already debited (to €500), but the travel agency's account is not yet credited (still is €10000). After the operations have been performed, i.e., the money is transferred, and the transaction commits, the result of the money transfer is durable: whatever failures occur, the result of this money transfer will always be represented in the system (requiring, for example, system recovery techniques with backups and logs).

3.4 Conclusion on classic history

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 transactions execute on only one database system. However, they lack the flexibility to meet the requirements of complex applications. For example, multi-database operations need a certain level of transparency for the interactions with each local database; a workflow system needs to support long-living transactions etc. Even though the flat transaction model with its ACID properties is not suitable for the complex application requirements, it is currently still the most used transaction model. The reason for this is the simplicity of the flat transaction model.

4 Middle ages: advanced transaction models

Although the flat transaction model is very simple and secure, it lacks the ability to support applications requiring long-living and/or complex transactions. Therefore, since the 'middle ages' of the transaction management history, advanced transaction models have appeared to address such needs.

The fundamental logic 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 again be divided if necessary. The advanced transactions can perform more complex and longer-lasting tasks. For instance, when a failure occurs during the execution of a long-living transactional process, the system is able to restart from the middle of the transaction instead of the very beginning.

4.1 Checkpoints and save points

Since databases usually contain a very large amount of data, a mechanism was required to support failure recovery, i.e., how to minimize the damage to a database in case a failure occurs. For this reason checkpoints were introduced. When creating a database checkpoint, the entire state of a database is made persistent, e.g. all operations stored in volatile memory (cache), are written to stable storage. In case a (drastic) failure occurs, the state of the database created at the checkpoint can be

restored, and operations performed thereafter can be redone using the information stored in persistent logs. The mechanism of checkpointing is still used in current database systems, such as Oracle Database [59], IBM DB2 [40], Berkeley DB [39].

The history of advanced transaction models can be traced back to the mechanism of save point, a concept first introduced in [2], which enables a transaction to roll back to an intermediate state. The authors suggest that during the execution of a transaction, a save point can be marked which returns 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 database to a previous state in case of failures.

When applying the rollback mechanism using save points, we should be careful about its constraints and limitations. 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 actually follows 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 a persistent save point has been proposed to overcome the deficiency, it is hard to implement this idea in reality [11]. For example, the database content can be rolled back to a previous state, but the local programming language variables will be lost. Another point to notice is that rollback is different from abortion. When aborted, the transaction is returned back to the state in which it started and the execution doesn't continue anymore. In contrast, a transaction which has rolled back to a save point still continues execution until it completes.

The save point mechanism led to later development of advanced transaction models, which emerged since the mid 1980s, e.g. distributed transactions, nested transactions, chained transactions. These models are more or less application specific and each of them addresses the need of 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. If 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 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 [30].

4.2 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, a Multi-database System (MDBS) imposes global integrity constraints on a transaction. Also a MDBS addresses other concerns like global atomicity and isolation. For instance, the whole transaction is aborted if any sub-transaction fails. In [12], a most popular model at that time, the 'base transaction model' is introduced.

The model defines two types of transactions: local transactions and global transactions. Local transactions are executed under the control of the local DBMS, while the MDBS is in charge of global transactions. Several approaches to realize transaction atomicity and database consistency are discussed. [12] also proposes the possible extensions to this basic model and provides an overview of the most recent work until then in the MDBS area and raises some open problems for future research such as a need for the standardization of operating system, communication interfaces and database systems.

The most influential work underlying distributed transactions is the X/Open Distributed Transaction Processing (X/Open DTP) model [83], a software architecture developed by X/Open, a consortium of vendors who are working on 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 [83]. The X/Open DTP model is a standard for the Two Phase Commit (2PC) protocol, a key technology, already developed at the end of the Seventies, ensuring agreed outcome between participants in a distributed transaction [31]. In the X/Open DTP model, the transaction manager is a functional component managing global transactions and coordinating the decision to start, commit or roll back, and ensures atomicity at a global level. Each participating resource manager is responsible for the ACID properties of its own resources.

In contrast to distributed transactions, which use a bottom-up approach to divide transactions into sub-transactions from a system topology point of view, nested transactions adopt a top-down method to decompose a complex transaction into sub-transactions or child transactions according to their functionalities. Moss [53] first discussed this concept by programming transactions in a structured way. As the work 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. 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 completed. The commit of a child transaction is conditional on the commit of its parent. Each child is atomic, thus it can abort independently regardless of its parent and siblings. When it aborts, the parent will take an action, for example triggering another sub-transaction as an alternative. The aborted sub-transaction results as if it had not executed. However, the aborted sub-transaction may have changed the state of the database so it can make the database inconsistent while the whole nested transaction still meets the consistency requirement. The mechanism of the model is very powerful and has a strong relationship with the concept of modularization in software engineering [30].

To illustrate nested transactions, consider the example shown in Fig. 3. The example concerns a travel agency application for selling holiday trips. In this ‘Middle Age’ era, the trip selling process would be implicit (hard-coded) inside the implementation of this application. However, for clarity, the left-hand side of the figure shows a process view of the application to illustrate the different functions inside the

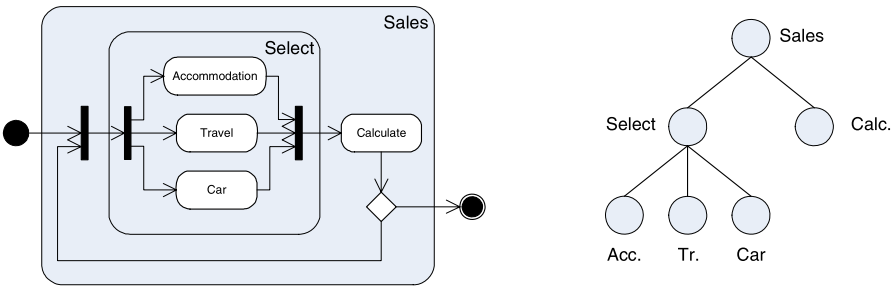

Fig. 3 Nested transaction model example

application. The right-hand side of the figure shows the process in a tree structure representation to illustrate the suitability of the nested transaction model.

Selling of holiday trips, consists of selecting an accommodation, a means of travel, and an optional rental car. Then after the total cost is calculated, the trip can be booked (another application, not shown in the figure) or changed. Selecting the three options can be done in parallel, but each selection depends on the other, for example, if the accommodation is located far from the beach, a rental car is required. Only after all three selections are made and the calculation is performed, should the result of the trip be made permanent (and thus visible to concurrent running transactions). Also, the total selection is only successful when all three selections are made and the sales is successful when the selection and calculation have succeeded. The nested transaction model support these requirement, as explained above. The *Select* 'function' determines the outcome and possible failure resolvement of the three separate selection 'functions' and the *Sales* 'function' determines that for the selection and calculation function.

Reference [80] proposes multilevel transactions (also called layered transactions) and their generalization, open nested transactions, based on the idea of nested transactions. A multilevel transaction is a variation of a nested transaction where a transaction tree has its levels corresponding to the layers of the underlying system architecture. Note that the leaf nodes are all at the bottom level, i.e. the depth levels of these leaves are the same. Reference [80] also introduces 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. Multi-level transactions differ with respect to nested transactions in three aspects [49]. First, children are executed only sequentially, not concurrently. Second, all the leaf-level sub-transactions are at the same 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.

When the structure of the transaction tree is no longer restricted to having all leaves at the same level (i.e., leaves at different levels are allowed), then multitransactions evolve into open nested transactions. How the ACID properties of open nested transactions are relaxed to achieve the ideal orthogonality, so that each of the ACID

properties can be omitted without affecting the others, is investigated to some extent in [80]. Compared to nested transactions that guarantee global level isolation, open nested transactions relax the isolation property in the global level to achieve a higher level of concurrency.

4.3 Chained transactions and sagas

Although the nested transaction and its extensions are more powerful than the classical flat transaction, they are only applicable in specific environments such as 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 [30], 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 traditionally each interval between every two save points is only part of an atomic transaction. In the chain, a sub-transaction triggers the next upon commit, one by one, until the whole chained transaction 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 in aborting 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 which include a compensation mechanism to roll back already completed subtransactions. The saga model described in [29] is a classic transaction model used as a foundation of many later transaction models. 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. 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 as a whole preserve application-dependent atomicity, meaning that the application logic determines what constitutes a failure in the saga and thus determines when a saga is fully compensated. So, as with ACID transactions, a saga transaction successfully completes or is completely rolled back (compensated). Similar to chained transactions, other transactions can be executed concurrently with a saga transaction. Because subtransactions of a sage commit on successful completion, their results become visible to other transactions even before the entire saga commits. Thus isolation is not guaranteed and

consequently, consistency in sagas is not realized by serializability, a common technique to keep the database consistent when accessed by multiple transactions. The saga model is an important transaction model which attracted a lot of attention. For example, some extensions of sagas are introduced in [16] with more recovery options.

4.4 Conclusion on the middle ages

Advanced transaction models can be viewed as various extensions to flat transactions that relax one or more ACID constraints to meet the specific requirements of complex applications. Two strategies exist to achieve different structures inside a transaction. One is to modularize a complex transaction with hierarchies. By this means, a large 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. These kind of transaction models are most suitable for multi-database systems and complexly structured applications. Another strategy is applied in chained transactions, sagas etc, through decomposing a long-lasting transaction into shorter sub-transactions. By means of splitting up 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 failure. These type of transaction models are therefore most suitable for long-running applications, i.e., applications with a long execution time span. An approach that combined both the component-transaction and sub-transaction strategies was developed in the next era and is known as the WIDE transaction model (see Sect. 5.5). It also combined concepts from the save point approach with concepts from the saga transaction model.

Note that there are other proposals that belong to this era. We skip the description of them because they do not exhibit an internal structure as typical as the above mentioned advanced transaction models. For example, Split-transaction [61] is proposed for open ended applications where the finish time is unknown in advance so it has a dynamic structure (split and join). The flex transaction model proposed in [24] for MDBS applications can also be viewed as an advanced transaction model consisting of an inexplicit hierarchy of local autonomous transactions, though it bears some characteristics of workflow transactions.

However, the abundance 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. From the transaction models covered in this section, the nested transaction model (in both open and closed form) and the saga transaction model have had the most influence: most of the transaction models that were developed in the eras that followed the ‘Middle Ages’ are based on them.

5 Renaissance: workflow transactions

The renaissance era saw the advent and rise of the Workflow Management Systems (WfMSs). The (research) focus in this era turned towards business process modeling,

execution and redesign. WfMSs made the automated execution of business processes possible, but did not offer any transactional support so that robustness, reliability and consistency were not guaranteed by the system.

Based on the advanced transaction models discussed in the previous section, specific transaction models have been designed for the support of business processes, usually identified as workflow transaction models or transactional workflows. Below, we first explain the concept of transactional workflow, which we use interchangeably with workflow transaction in this paper. Next, we review the ConTract model, an early attempt in this area which is not specifically intended for workflow transactions but in general for long-lasting and complex applications. Then we cover three transaction models developed to support complex workflow processes (the first two focus on intra-organizational processes, the third focuses on inter-organizational processes), which are a good representation of the research done in the area of workflow transactions.

5.1 Transactional workflow versus workflow transaction

The concept of transactional workflow was first introduced in [68] to clearly state the relevance of transactions to workflows. Since the mid 1990s, two developments have taken place in the area of workflow technologies. One is the development of transaction models supporting workflows (workflow transactions) and the other is the development of languages for workflow specification (transactional workflows).

From a transactional point of view, workflows are generalized extended transactions which 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 thorough analysis of transactional workflows and workflow transactions can be found in [34].

5.2 Activities/transaction model

One of the earliest approaches in transactional workflows is introduced in [21, 22], in which a transaction model for long-running transactions, called ATM (Activities/Transaction Model) is presented. In ATM, each step of an activity is modeled by a transaction.

At first, the control flow among the steps is expressed implicitly by rules. The rules are triggered by an event or by the rule system detecting that some specified event has occurred in the database. In the extension of the original model [21] the control flow and data flow between the steps of an activity may be specified statically in the activity's script. A proposed architecture is based on a simple nested transaction implementation and uses the services of a reliable queuing facility by which the atomic steps are connected.

5.3 ConTracts model

First proposed in [64] and finalized in [79], the ConTract model addresses the transactional challenge for long-lasting and complex computations in a formal basis and is aimed at the problem domain of large distributed applications. Although the ConTract model is not labeled as a workflow transaction model it is applicable in the workflow transaction area, because:

1. it uses workflow concepts. For example 'step', 'flow', and 'script'
2. its basic idea is to model control flow by programming short ACID transactions into large applications thereby guaranteeing reliability and correctness along the execution.

Unlike the advanced transaction models mentioned before, the ConTract model does not extend the ACID transactions in structure but embeds them in the application environment and provides reliable execution control over them. It defines a unit of work as a step which ensures the ACID properties but preserves only local consistency. These steps are executed according to a script, which is an explicit control flow description. A reliable and correct execution of the steps is called a ConTract. The ConTract model offers control mechanisms like semantic synchronization, context management and compensation at the script level to provide transaction support to a long-lived and complex application.

5.4 Exotica

The approach developed in the Exotica project is based on compensation of workflow processes [1]. Within this project, transaction support for workflow processes was developed as an extension to the Flowmark workflow management system. The Exotica approach relies on statically computed compensation workflow patterns that are used as extensions to the basic workflow specification. This means that all possible compensation routes for a specific process are determined, which are then 'infused' into the normal process specification. An obvious drawback is that the resulting process specification becomes very complex. Even though this specification is only used by the machine, the necessity exists that two process models/specifications are required, one for a business modeler (an actual user of the system) and one to be processed by the machine. This complicates maintenance and evolution of workflow process models. Another drawback is that the approach does not handle loops (iteration) in the process model.

The FlowBack software is a commercial implementation based on this approach [45].

5.5 The WIDE transaction model

In [36], a two-layer transaction model, known as the WIDE transaction model, is presented. The model combines the concept of safe point with Sagas, so that more flexibility is offered in compensation paths in case of exceptions. Note the difference between 'save' point and 'safe' point. The former is a database operation in which

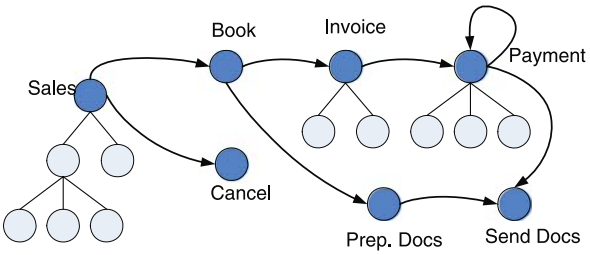

Fig. 4 WIDE transaction model example

(part of) the database is made persistent (saved to stable storage), while the latter is a point in a business process that is considered to be safe, i.e., the business process execution is known to be in a consistent state.

In the WIDE transaction model, the bottom layer consists of local transactions with a nested structure that conform to the ACID properties, based on the nested transaction model described in Sect. 4.2 [13]. The upper layer is based on an extension of Sagas that roll back the completed sub-transactions using the compensation mechanism, thus relaxing the requirement of atomicity [74]. The semantics of the upper layer have been formalized using simple set and graph theory [35]. The local transaction layer is designed to model low-level, relatively short-living business processes, whilst the global transaction is designed to model high-level long-living business processes.

Figure 4 shows a high-level view of a travel booking process, used to illustrate the WIDE transaction model. The entire travel booking process is a long-running process that can last for several days or weeks, while some of the activities themselves have a complex internal structure. The WIDE transaction model is very suitable for such circumstances. Its two layers satisfy the long-running process requirement as well as the complex structured activities. In the figure, the dark colored circles represent the activities (or sub-transactions) supported by the upper layer, while the three activities, i.e., *Sales*, *Invoice*, and *Payment* with an internal nested structure are supported by the lower layer of the WIDE transaction model. Suppose the *Book* activity is considered a safe point, then a failure in the process execution after that activity will result in a rollback using compensating activities to (not including) the *Book* activity, after which the process could continue its execution again.

Failures within the lower layer of the WIDE transaction model are resolved in a similar manner as explained in the example shown in Fig. 3 illustrating the nested transaction model, see Sect. 4.2.

The flexible approach of the WIDE transaction model is adopted later in [73] to develop the X-transaction model, see next subsection. Note that these two models address needs in different contexts. The WIDE transactional model caters for intra-organizational workflows while the X-transaction model offers support for cross-organizational (also called inter-organizational) workflows.

5.6 X-transactions

The X-transaction model is a three-level, compensation-based transaction model to support cross-organizational workflow management. It has been developed in the

CrossFlow project [28], where a contracted service outsourcing paradigm was proposed [73]. The three levels in this model are the outsourcing level, the contract level and the internal level, each with a different visibility to the consumer or the provider organization.

The model views an entire cross-organizational workflow process as a transaction. The involved intra-organizational processes are divided into smaller I-steps that adhere to ACID properties. Each I-step has an associated compensating step that is used to roll back (part of) the process in case of failures. Similar to this idea, the contract-level cross-organizational process is divided into X-steps, each of which corresponds to one or more I-steps. As with I-steps, compensating steps are associated with X-steps and used in case of failures, which, in this case, involve the cross-organizational process (and thus involve both parties, i.e., the consumer and provider organization). 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 combinations of rollback scopes and rollback modes.

An architecture to support this model is also presented in [73]. The architecture consists of three layers. The top layer supports the cross-organizational aspect of the model. Its components are dynamically created whenever (part of) a process needs to be outsourced. These components facilitate the searching of a suitable partner (provider organization), the creation of an electronic contract and the execution of the outsourced process, including required cooperative support services, like transaction management, level of control, and quality of service monitoring [28]. After the service outsourcing has finished (end of contract), these components are destroyed again. The middle and bottom layer of the architecture are static (always available). The bottom layer supports the execution of intra-organizational business processes and therefore includes the workflow management system(s), as well as transaction support. The middle layer isolates the bottom layer from the 'outside world' to provide portability with respect to specific WfMSs.

5.7 Conclusion on the renaissance

The focus of workflow applications is the control-flow, which is different from the data-centric database applications. From the above discussion, we can see that transaction support for workflows is not restricted to ensure the ACID properties anymore. Workflow transactions usually leverage the traditional transaction mechanisms for recovery and concurrency control but meanwhile address more coordination requirements. A limitation of workflow transaction is however the platform/software dependency.

The transaction models in this section are covered in a chronological order. Some of the newer transaction models therefore extend and improve upon the older models. For example, the X-Transaction model improves upon the WIDE transaction model by adding transaction support for inter-organizational processes (instead of only supporting intra-organizational processes). The WIDE transaction model improves upon the Exotica approach through the dynamic creation of compensating processes instead of statically calculating and integrating all possible compensating process possibilities. It also added support for loops in processes executions.

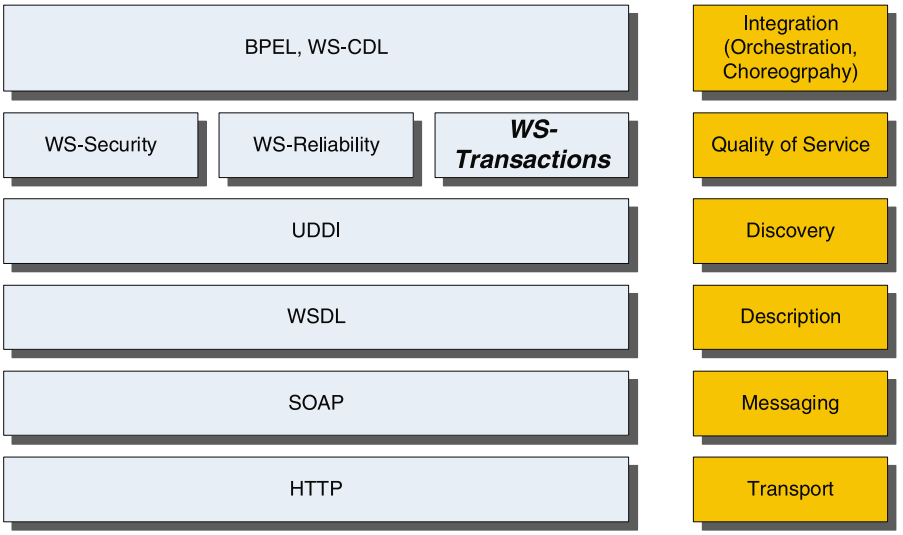

Fig. 5 Web services stack

With the development of Internet applications, transaction support for today's workflow processes has also evolved to put more attention to communication, distribution and coordination aspects. In the next era the focus moved from workflow processes to services. Both processes and services are basically a dual view on the same 'thing' [75]. Along with this trend, we discuss transaction management in the Internet environment (with a focus on Web Services) in the next section.

6 Modern times: web services transactions

From the late 1990s until now, the area of transactions in the loosely-coupled Web services (WS) world has received more and more attention. In addition to other accepted standards such as SOAP [67], WSDL [81], UDDI [72] etc., a technique to guarantee the consistency and reliability of WS applications is needed. Three competing standardization efforts have been taken in this era, which are described and compared in the following subsections. Currently the WS-Transaction effort (see Sect. 6 has become the de facto standard for Web Services transaction support (it also became an OASIS standard), see also Sect. 6.4.

Figure 5 shows the Web Services stack to clarify the position of the Web service transaction area amongst the numerous other Web service areas. As can be seen in the figure, Web Services transactions are part of the Quality of Service aspect specified for Web services.

6.1 Business transaction protocol

The Business Transaction Protocol (BTP) [14], developed by OASIS, is, as its name indicates, 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. BTP ensures consistent outcomes for parties that use applications that are disparate in time, location and administration and participate in long running business transactions [50].

In a BTP compliant Web service environment, Web services are the software agents that contain business logic and handle request and response messages. Web services usually use backend systems to store data, however, the Web service themselves are not in control over the backend systems and therefore cannot influence the outcome of a request to the backend system. This is under the control of the transaction manager. Backend systems can have a role as participant within a transaction if they are 'wrapped' as a Web Service, i.e., the backend system acts as a Web Service. This enables communication between the transaction manager and the backend systems (via the web service interface (ports) specified for the backend system).

BTP is a protocol with two distinct phases, geared towards the outcome of the transaction. Every phase of a transaction within BTP stands on its own and may be implemented in any way by a BTP compliant Web service (or application). In the first phase of a transaction, the BTP participants are instructed to make provisional or tentative state changes (called the provisional effect). In the second phase the participants are instructed to complete the transactions either through confirmation (called the final effect) or through cancellation (called the counter effect). How the mentioned effects are implemented is fully dependent on the participants. Counter effect for example could be implemented by compensation techniques or by traditional database rollback approaches.

To reflect the differences with the traditional 2PC protocol, the commands used in BTP are different and extended from 2PC, thereby also abstracting from specific background implementations. In phase one of the BTP protocol, 'prepare' is used to ask the participants to make the provisional effect and in phase two, 'confirm' and 'cancel' are used to ask for either the final effect or the counter effect. BTP therefore guarantees atomicity of decision [23] (i.e., ensuring that a valid transaction is conducted) whilst at the same time abstracting from implementation specific details.

What makes BTP unique in comparison with the traditional 2PC protocol is that in BTP, the application controls the time between the two phases. In traditional 2PC, the second phase of the protocol starts immediately after the first phase ends. In BTP, the application, using business logic (knowing the semantics of the application), can determine when to start the second phase of the protocol. Also, instead of the transaction manager, it is the application that can determine which participants to commit (called a 'consensus group') and which to cancel.

BTP specifies two extended business transactions types [14]:

1. Atoms. If a consensus group is specified as an *atom*, it is guaranteed that the transaction outcome of this consensus group is atomic, meaning that either all participants confirm or all participants cancel.
2. Cohesions. If a consensus group is specified as a *cohesion*, the atomicity property of the group is relaxed (compared to an atom). The application itself determines (using business logic) which participants to confirm or cancel.

Participants of a cohesion that are confirmed, i.e., should commit their results form a 'confirm set'. The confirm set itself is in turn an atom, as all members of this set should complete successfully (they are confirmed). Cohesions are used to model long running transactions in which participants enroll in atoms which may be canceled or prepared, depending on certain conditions. It may take considerable time (several hours or days) for the cohesion to arrive at a confirm set.

6.2 Web services transactions

The second candidate is the combined Web Services Transactions specifications, collectively indicated by WS-Tx, consisting of WS-Coordination (WS-C), WS-AtomicTransaction (WS-AT), and WS-BusinessActivity (WS-BA) [55–57].

The 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. The specifications are aimed at the reliable and consistent execution of web based business transactions using different interconnected Web services.

6.2.1 WS-coordination

A lot of issues related to Web services require coordination, for example security, transaction management, replication, work flow, and caching. As opposed to BTP, in which coordination is interwoven with transaction management, WS-Coordination (WS-C) defines a framework that solely focuses on outcome determination and processing. This way WS-C provides a generic coordination infrastructure for Web services, making it possible to plug in specific coordination protocols [26, 52].

The coordination framework specifies three services necessary to support coordination protocols.

- The Activation Service, which creates a new activity coordinator that uses a specific coordination protocol for a particular application instance.
- The Registration Service, which is responsible for the enrollment of new participants and selection of the coordination protocol.
- The Coordination Service ensures that the registered Web services are driven through completion by using the selected protocol. The protocol defines the behavior and the operations that are required for the completion of an activity.

Currently, the WS-Transaction specifications (WS-AT and WS-BA) are the first and only protocol specifications based on WS-Coordination.

6.2.2 WS-AtomicTransaction

The WS-AtomicTransaction specification is focused on existing transaction systems and protocols with strict ACID requirements. Existing transaction systems that require an all-or-nothing outcome, form an important part of the companies' backend infrastructure. Traditionally, these systems are heterogeneous and coupling these systems together within one organization is the first step towards interoperability.

However, enabling interoperability between systems of different organizations is not just a matter of coupling systems, as this introduces the well-known interoperability problems related to semantic differences, autonomy, non-disclosure, etc. Presenting the backend systems/applications as services, i.e. wrapping them as a Web Services, is a valid solution for most of the problems mentioned.

The WS-AT specification introduces an extension to the Web Services Description Language (WSDL), with which it is then possible to offer an application as a Web Service, but also to specify its transactional behavior (ACID behavior) in case the service is involved in an intra-organizational process that assumes the WS-AT transaction protocol.

Participants in an WS-AT should support the following requirements that are assumed by the protocol [57]:

- Updates are isolated until they are committed.
- There is a single all or nothing decision for all participants.
- Any participating systems in the transaction can abort the entire transaction.

This requires a high mutual trust between the participants in such a transaction, making it hard to use this protocol for inter-business transactions.

The following protocols are specified in WS-AT: Completion, TwoPhase Commit (2PC) with two variants, Volatile 2PC, and Durable 2PC. Details of these protocols can be found in the WS-AT specification [57].

Figure 6 shows an example scenario in which WS-AT is used. This example is (again) a travel booking process, with the *sales* part modified compared to the example given in Sect. 4.2. Here the three selection activities are replaced by Web services, each dealing with the specific selection topic. These three services are participants in a WS-AT transaction to guarantee that all three selection services successfully complete (commit) or none of the three. Details on the transaction specification of these services and the protocol execution with the exchanged messages are rather verbose and therefore left out of this paper, but similar details can be found in the WS-Tx specifications [7, 57] or in [60].

6.2.3 WS-BusinessActivity

As the WS-AT specification resembles very much traditional 2PC ACID transactions, it suffers from the same problems. To circumvent these problems and to support long running business transactions, the business activity coordination type has been introduced [56].

As seen before, atomic transactions handle system generated exceptions transparently from the application that drives the transaction. The application that drives the business activity, which spans multiple atomic transactions and tries to move from one consistent state to the other, therefore does not need to care about those system generated exceptions, but can focus on the handling of application generated exceptions (i.e., business exceptions). To handle these kinds of exceptions and the overall coordination, business logic within the application that drives the overall business transaction is required. WS-BA uses atomic transactions to preserve the autonomy of participating organizations whilst at the same time providing mechanisms to reach overall agreement.

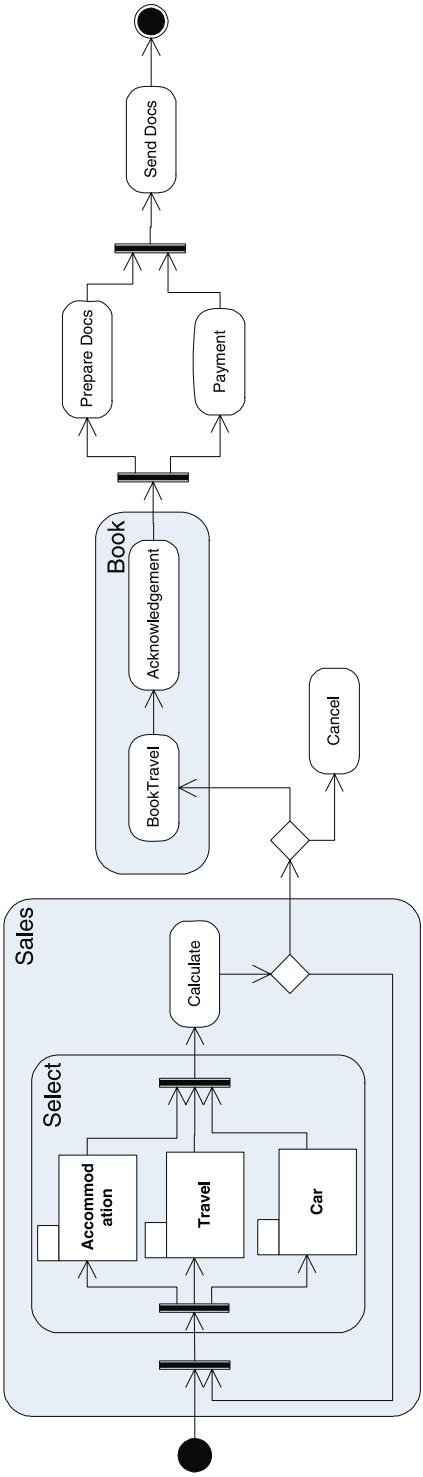

Fig. 6 WS-AT example

Some characteristics of business activities in WS-BA are:

- Business activities may be partitioned into hierarchical nested scopes. Compensating actions may be registered with the parent activity to undo completed child tasks. Exception handlers make use of application logic so that the overall business activity can continue (i.e., forward recovery).
- Results of completed tasks (e.g., transactions) within a WS-BA can be seen prior to the completion of the business activity, thereby relaxing isolation.
- Participants are autonomous and can exit activities at will, thereby delegating processing to other scopes (participants) or exiting without knowing the outcome of the protocol.
- A participant may specify its outcome directly to the coordinator without being asked for it. In case the outcome is negative (participants activity failed), the exception handler can take this into account and adapt the business activity to circumvent this failure.
- The state of the business activity is made persistent between steps in order to reach a desired goal, even if exceptions occur.

The WS-Business Activity specification defines two coordination types for a coordinator, namely the atomic outcome type and mixed outcome type. The first type requires the coordinator to drive all participants to the same final state. The latter type allows a coordinator to choose which participants need to commit or compensate. The behavior of the coordinator is determined by the application driving the activity. Besides the two coordination types, the following two coordination protocols are specified: BusinessAgreementWithParticipantCompletion and BusinessAgreementWithCoordinatorCompletion. The reader is referred to the specification for details on these protocols [56].

The WS-BA specification does not specify how the 'knowledge' of the coordinator is communicated to the application that drives the business activity, so that it can make decisions based on that knowledge. This is left to the implementers of the specification. The responsibility of the coordinator is to ensure that participants and the coordinator itself are in the same state. The WS-BA specification contains state tables that specify the behavior for certain circumstances, e.g. how to handle duplicate messages.

6.3 Web services composite application framework

The third Web Services transaction candidate is the Web Services Composite Application Framework (WS-CAF) [8]. The purpose of WS-CAF is to provide an interoperable, 'easy-to-use' and 'easy-to-implement' framework for composite Web Services applications. It is composed of a series of specifications: WS Context [6], WS Coordination Framework [7] and WS Transaction Management [9]. Each specification covers a certain level of the overall architecture required to build reliable business applications that span multiple systems and use Web service technology.

6.3.1 Web services context

The Web Service Context specification [6] defines a generic context management mechanism for sharing common system data (i.e., context) across multiple Web ser-

vices [54]. This in contrast to the context specifications in WS-Tx and BTP, where the context is combined with coordination (WS-Coordination in WS-Tx) or combined with coordination as well as transaction management (BTP).

However, context information is not just relevant for transactions, for example, Web services security relies heavily on context exchange. Also correlation of different Web services that share certain data (ID's, tokens, etc.) to establish relationships between them is of importance within web applications. The WS Context specification (WS-CTX) [6] provides for this functionality making it possible to connect multiple Web services into one activity (scope) by correlating them using specific context information that is not managed by a coordinator.

6.3.2 Web services coordination framework

The second level in the WS-CAF specification is the Web Service Coordination Framework (WS-CF) which provides a coordination service that is plugged into WS-CTX. It manages and coordinates multiple Web services that are grouped together in one or more activities to perform some task together.

WS-CF supports, just like WS-C, different coordination protocols. However, the specification of WS-CF is more thorough than WS-C. Also, the architecture is specified very exhaustively with respect to the requirements imposed on the participants of a coordinated activity. The WS-CF architecture has three main components.

1. The Coordinator, at which participants can register so that they receive the context and outcome of an activity.
2. The Participant of an activity, which offers operation(s) that are executed during the coordination sequence processing. The role of coordinator and participant is used to define the protocols and related messages between them.
3. The Coordination Service, which defines the behavior for a specific coordination model. The coordination service provides a processing pattern that is used for outcome processing [6]. How the service is implemented is not defined in WS-CF, but depends on higher level protocols (of which WS-Transaction Management (WS-TXM) defines several related to transaction management).

When using coordination within an activity, there will be two distinct message flows. One application message flow between participating Web services and one coordination message flow between the coordinator and the participants to handle context propagation and exception handling [54].

6.3.3 Web services transaction management

The Web Service Transaction Management (WS-TXM) specification defines three protocols that are plugged into WS-CF [9]. These protocols can be used together with a coordinator to negotiate a set of actions for all participants that are to be executed based on the outcome of a series of related Web services executions [8]. To reach an agreement of outcome among the participants of a transaction in a consistent way, the transaction protocols can use context information and coordination protocols. The Web service executions are linked together in scopes by the overall context and can

be nested and executed concurrently [52]. WS-TXM binds the scope of an activity to the lifetime of a transaction.

Three specific transaction models are defined in WS-TXM, which can be used for different situations:

1. ACID transaction. This model resembles the traditional ACID transactions but applies it to Web services, enabling tightly-coupled network-based transactions, which are most suitable (just like WS-AT) to achieve interoperability between existing transaction systems within one organization.

   Participants of an ACID transaction (registered at a coordinator) can only be removed when the transaction terminates (either successful or unsuccessful). The coordinator uses the 2PC protocol to control the registered participants, but some optimizations to this protocol are available.
2. Long Running Action (LRA). This model is designed to cover transactions that have a long duration. In this model, an activity is seen as a set of business interactions for which compensations should be possible (comparable to Sagas, see Sect. 4.3). How services compensate is implementation specific and depends on the activity itself.

   The LRA model only defines a protocol identifying the trigger for compensation actions and the conditions under which the triggers come into effect. Similar to ACID transactions, LRAs are bound to the scope of the activity. If the work done within an activity is finished, the termination will cause the LRA protocol to be executed. The protocol will either accept or compensate the work. In contrast to the ACID transactions, the LRA protocol defined by WS-TXM is a one-phase protocol.

   The LRA model allows nesting. But for the enclosing activity it must be possible to compensate the work of the nested activity until it terminates as well. Recovery mechanisms (like for example checkpointing) defined in WS-CF can be used to recover participants. Services can register as a compensator at the coordinator. The compensator can be instructed by the coordinator to undo the work performed by a service or to compensate work that could not be completed by a service. Compensation may also be carried out by other LRAs (which can also have compensators). A compensator can be invoked at any point in time during a LRA by a coordinator.
3. Business Process transaction model (BP model). The aim of this model is to integrate different heterogeneous transaction systems (e.g., using ACID transactions and messaging) from different business domains into one overall business to business transaction [52].

   The business process is managed and controlled by an overall business transaction manager (i.e., the coordinator) that communicates (a)synchronously with the individual tasks. If all tasks have completed, the business process can terminate in a confirmed manner. If not, it will terminate in a canceled manner and all the work performed in the process will be rolled back. In the BP model the failures that can occur are handled either by replay, void, compensation or human intervention.

   Interposition is an important concept in the BP model. Every domain (party), which has its own transaction systems and protocols, is hidden behind an interposed coordinator. This interposed coordinator translates the BP protocol messages into the specific protocols (e.g., like ACID, BTP, LRAs) used within the

domains. The root coordinator (the business transaction manager) has a parent-child relationship with the subordinate coordinators. This coordinator keeps track of the domains and the state they are in. The exact definition of a domain and what services belong to a domain is not covered in the specification.

Depending on the situation in which the business process is to take place, and the specific transactional requirements of the business process, a suitable transaction model is available from the WS-TXM specification.

The next subsection presents a comparison of the three different web services transaction specifications, i.e., BTP, WS-Tx, and WS-CAF.

6.4 Conclusion

In [52], a comparison between BTP and WS-Tx is given. 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. In 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 underlies 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 [46]. This technical report gives a detailed overview of the three specifications and highlights the differences between the three candidates. In 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. However, since then, the WS-Tx specification has become the de facto standard (supported by the OASIS standardization organization) for Web service transactions.

In [42], 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. As already stated above, the WS-Tx is the 'winner' of the three Web service transactions standardization efforts. The BTP standardization effort has since been closed.

7 Modern times: grid transactions

Grid computing is a form of distributed computing that involves coordinating and sharing computing resources across the web globally, thereby making the exclusive immense computing power previously only available to a few organizations now available to everyone. This emerging technology has been gaining a lot of attention, however, the focus is mostly towards infrastructure and middleware. Less effort is spent in the area of Grid transactions. Below, we describe a few efforts currently ongoing in this area.

7.1 Global grid forum

The 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 [70] 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 Sect. 6.

7.2 GridTP

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 [62]. The authors claim that GridTP provides a consistent and effective way of making existing autonomously managed databases available within Grid environments.

7.3 Unnamed grid transaction approach

Some of the significant questions that arise when applying transactions into a grid environment are tackled in [71]. A protocol ensuring correct execution of concurrent applications on 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 between dependent transactions and the peers they have accessed. The idea is innovative in the sense that it combines old concepts and techniques for a new purpose.

7.4 Conclusion on modern times

It is hard to evaluate and compare the above approaches at this early stage of this relatively new area. We can expect that with the development of the Grid technology,

the need for a standard protocol to provide transactional support will result in more and more effort as Grid computing does require a reliable way to coordinate and communicate. In what way, if (part of) the results from the Web Services transaction area can be reused remains to be seen at this moment.

8 Transaction frameworks

Besides the transaction models covered in Sects. 3 to 7, transaction frameworks exist that encompass more than a single transaction model. This section covers some of these frameworks.

We first present two conceptual transaction frameworks, followed by two middleware technical transaction frameworks.

8.1 Conceptual transaction frameworks

The following subsections discuss two transactional frameworks defined on a conceptual level, one older (ACTA) and one new (BTF).

8.1.1 Meta transaction model (ACTA)

A novel approach was used in [15], in which a comprehensive framework named ACTA is developed, by unifying existing models to capture the semantics and reason about the concurrency and recovery properties of complex transactions. Later, more elaborate extensions to this ACTA model were described in [16–18].

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 can have on other transactions are specified as ‘commit-dependency’ and ‘abort-dependency’. Commit-dependency describes the relationship of one transaction T1 to another transaction T2. This dependency rule regulates that T1 cannot commit until T2 either commits or aborts. Abort-dependency describes the relationship of T1 to T2, which means if T2 aborts, 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 objects contained in a ‘view set’ or ‘access set’. The view set contains all the objects potentially accessible to the transaction while the access set contains the objects that have already been accessed by the transaction. Transactions make changes to 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 combining delegation mechanism with commit and abort dependencies, it is possible to 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.

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 [10], which uses primitives at a programming language level based on ACTA building blocks such as 'history', 'delegation', 'dependency', 'conflict set'.

8.1.2 Business transaction framework

The XTC (eXecution of Transactional Contracted electronic services) project aims at laying a generic foundation to the transactional support for processes in a service-oriented environment. Within this project, a Business Transaction Framework (BTF) is being developed to support contract-driven, inter-organizational business processes. The basic idea of the BTF is to extract and group existing transaction models into an Abstract Transaction Construct (ATC) library and select the required ATCs to compose a transaction hierarchy on demand [78].

The architecture of the Business Transaction Framework is a multi-level, multi-phase design [48]. Three phases exist along the BTF life-cycle; definition phase, composition phase and execution phase. During the definition phase, the ATCs are abstracted from the classic and widely-adopted transaction models based on a taxonomy, which covers and classifies the existing work in transaction management domain. After the design of an ATC library, one can easily make use of these constructs to build a transaction plan for a complex process within the composition phase. The BTF also offers the flexibility to adjust the transaction plan to accommodate changes in business processes that can occur (quite frequently) in a dynamic business environment. The abstract plans resulting from the composition phase are instantiated to form concrete, executable business transactions, which can then be executed during the execution phase [47, 76].

The BTF has two distinctive features. First, it achieves flexibility by leveraging the existing models in the transaction management domain and abstracting them as the building blocks with which to construct transaction plans that support business processes. Second, it proposes to use contractual agreements to specify transactional qualities for today's business processes therefore business trustworthiness is guaranteed [82].

8.2 Middleware transaction frameworks

The following two subsection cover two middleware transaction frameworks that are defined within two well-known software platforms: CORBA and Java 2 Enterprise Edition (J2EE).

8.2.1 CORBA activity service framework

The CORBA Activity Service Framework has been developed to support advanced transactions (also called extended transactions) from within the middle-

ware layer [37]. It thereby complements the CORBA Object Transaction Service (OTS) [58].

Because the ACID transaction model is too restrictive in some application domains and it is unmanageable to 'hardwire' specific transaction mechanisms in the applications themselves, the CORBA middleware layer was extended. The extension, implemented as the CORBA Activity Service Framework (ASF), offers an event signaling mechanism with which it should be possible to enable activities to coordinate amongst each other in such a way that the extended transaction model required is simulated.

Using specific implementations of the framework, different extended transaction models are supported and can be mapped to these implementations. A specific implementation contains a *signal set* and associated *actions*. A signal set contains those signals needed to simulate (or represent) an extended transaction model. The specific implementation contains the protocol specification for the transaction model simulated and generates the required signals that are sent to the actions (i.e. activities in workflow terms) and processes the returned result to determine which signal to send next.

Thus the framework does not need to know the semantics (and operations) of an activity, but merely manages the coordination between activities following a certain protocol that represents an extended transaction model.

8.2.2 J2EE transaction framework

The Java 2 Enterprise Edition (J2EE) Transaction Framework started as a fundamental element within the J2EE Architecture [5]. It was developed to offer transaction primitives to developers using J2EE, including the Java Transaction API (JTA). Transactions in the framework adhered strictly to the ACID properties, albeit in a distributed component-based environment (using the Two-Phase Commit Protocol for example).

Two ways of specifying transactions exist:

1. Pragmatic. The application developer is responsible for delineating the beginning and ending of a transaction in the application.
2. Declarative. A component as a whole can be marked as being transactional, which means that the system itself is responsible for ensuring correct transactional execution of that component.

Realizing that the ACID properties are too limited in modern, advanced applications, the framework was (and is) extended further. For example the Java Activity Service [41] offers support for extended transaction models using similar mechanisms (SignalSets) as described for the CORBA Activity Service Framework, see previous subsection.

8.3 Conclusion on transaction frameworks

Two different approaches towards a transaction framework exist. The first approach treats transaction models on a conceptual level, thereby abstracting from specific

technicalities of those transaction models. In ACTA, transaction models are analyzed based on the dependencies between transaction operations. The resulting framework can then be used to create new transaction models as well. The focus of the Business Transaction Framework is on the abstraction of transaction models (ATCs) so that transactional properties can be identified that can subsequently be applied to support transactional requirements of business processes. New transaction models can be created in the BTF, i.e., by combining different ATCs, as well, but that is not a main goal of the BTF.

The second approach takes a bottom-up view; creating a mechanism, e.g., signals in CORBA ASF, that should then suffice to support existing and future transaction models as long as they can be mapped onto that specific mechanism. The usefulness of this approach needs to be proven in practice, as a specific implementation (of the signal sets and actions) is required for each (advanced) transaction model. A limitation of the approach is the coupling of coordination of activities and the transaction model, i.e., the coordination of activities is prescribed by the transaction model. This limits the suitability of this approach within the business process domain, in which routing (coordination) of activities is prescribed by the business process specification and not by a transaction model (which should merely support the execution of the business process).

9 Conclusions

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.

The evolution and relationships between (most of) the transaction models covered in this paper is depicted in Fig. 7. This figure makes clear if and on which other transaction model a specific transaction model is based. For example, the WIDE transaction model is based on the Saga model, which in turn is based on the (abstract) Chained transaction model. The right-hand side of the figure positions the transaction frameworks in the specific ‘eras’ and also shows that the Java Transaction Framework is based on the CORBA Activity Service Framework.

All transaction models positioned in the ‘Modern Times’ era have two other transaction models as a basis. The Web Services transaction models (as well as the GridTP approach) have their roots in the 2-Phase Commit (2PC) protocol/model as well as the Saga transaction model (compensation based support for long-running transactions). The grid transaction approach by the GGF in turn is based on the Web Services transaction models.

We notice that, with the development of information technology towards 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.

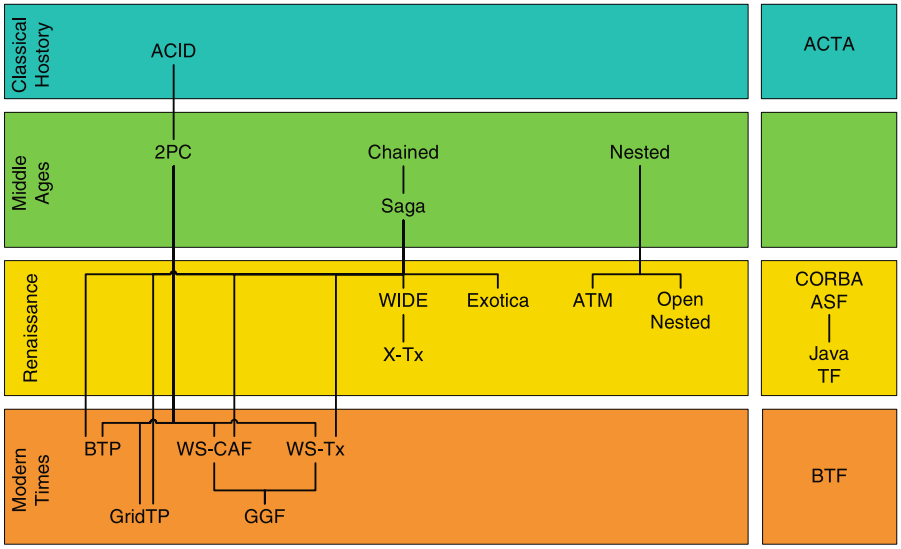

Fig. 7 Relations in transaction models

More concretely, we see that the Web service transactions and the mobile transactions might converge as they are both supporting thin, small clients (Web services and mobile device respectively) which represent autonomous 'parties' of which the accessibility cannot be determined beforehand. Also the area of process and service compositions requires the support for transaction composition. This means that it should become possible to compose transaction models in such a way that the transaction requirements of each (part of) a process can be satisfied by a component from the composed transaction. These are the challenging research areas that should and will be addresses in the future.

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

1. Alonso, G., Agrawal, D., Abbadi, A.E., Kamath, M., Günthör, R., Mohan, C.: Advanced transaction models in workflow contexts. In: Su, S.Y.W. (ed.) International Conference on Data Engineering (ICDE), pp. 574–581. IEEE Computer Society, Los Alamitos (1996)
2. Astrahan, M.M., Blasgen, M.W., Chamberlin, D.D., Eswaran, K.P., Gray, J., Griffiths, P.P., King III, W.F., Lorie, R.A., McJones, P.R., Mehl, J.W., Putzolu, G.R., Traiger, I.L., Wade, B.W., Watson, V.: System r: relational approach to database management. ACM Trans. Database Syst. **1**(2), 97–137 (1976)

3. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services: Concepts, Architectures and Applications. Springer, Berlin (2004)
4. van der Aalst, W.M.P., van Hee, K.: Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge (2004)
5. Baksi, D.: J2EE transaction frameworks: Building the framework. ONJava.com (2001). http://www.onjava.com/pub/a/onjava/2001/04/26/j2ee.html
6. Bunting, D., Chapman, M., Hurley, O., Little, M., Mischkinsky, J., Newcomer, E., Webber, J., Swenson, K.: Web service context (WS-CTX), July 2003. http://www.oasis-open.org/committees/ws-caf/
7. Bunting, D., Chapman, M., Hurley, O., Little, M., Mischkinsky, J., Newcomer, E., Webber, J., Swenson, K.: Web service coordination framework (WS-CF), July 2003. http://www.oasis-open.org/committees/ws-caf/
8. Bunting, D., Chapman, M., Hurley, O., Little, M., Mischkinsky, J., Newcomer, E., Webber, J., Swenson, K.: Web services composite application framework (WS-CAF), July 2003. http://www.oasis-open.org/committees/ws-caf/
9. Bunting, D., Chapman, M., Hurley, O., Little, M., Mischkinsky, J., Newcomer, E., Webber, J., Swenson, K.: Web services transaction management (WS-TXM), July 2003. http://www.oasis-open.org/committees/ws-caf/
10. Biliris, A., Dar, S., Gehani, N.H., Jagadish, H.V., Ramamritham, K.: Asset: a system for supporting extended transactions. In: Snodgrass, R.T., Winslett, M. (eds.) SIGMOD Conference, pp. 44–54. ACM, New York (1994)
11. Bernstein, P.A.: Principles of Transaction Processing. Morgan Kaufmann, San Fransisco (1997)
12. Breitbart, Y., Garcia-Molina, H., Silberschatz, A.: Overview of multidatabase transaction management. VLDB J. **1**(2), 181–239 (1992)
13. Boertjes, E., Grefen, P.W.P.J., Vonk, J., Apers, P.M.G.: An architecture for nested transaction support on standard database systems. In: Quirchmayr, G., Schweighofer, E., Bench-Capon, T.J.M. (eds.) Database and Expert Systems Applications (DEXA). Lecture Notes in Computer Science, vol. 1460, pp. 448–459. Springer, Berlin (1998)
14. Ceponkus, A., Dalal, S., Fletcher, T., Furniss, P., Green, A., Pope, B.: Business transaction protocol version 1.0, June 2002. http://www.oasis-open.org/committees/business-transactions/
15. Chrysanthis, P.K., Ramamritham, K.: ACTA: a framework for specifying and reasoning about transaction structure and behavior. In: SIGMOD Conference, pp. 194–203 (1990)
16. Chrysanthis, P.K., Ramamritham, K.: ACTA: the SAGA continues. In: Database Transaction Models for Advanced Applications, pp. 349–397. Morgan Kaufmann, San Mateo (1992)
17. Chrysanthis, P.K., Ramamritham, K.: Delegation in ACTA to control sharing in extended transactions. IEEE Data Eng. Bull. **16**(2), 16–19 (1993)
18. Chrysanthis, P.K., Ramamritham, K.: Synthesis of extended transaction models using ACTA. ACM Trans. Database Syst. **19**(3), 450–491 (1994)
19. Date, C.J.: An Introduction to Database Systems, vol. I, 4th edn. Addison-Wesley, Reading (1986)
20. Dunham, M.H., Helal, A., Balakrishnan, S.: A mobile transaction model that captures both the data and movement behavior. Mobile Netw. Appl. **2**(2), 149–162 (1997)
21. Dayal, U., Hsu, M., Ladin, R.: Organizing long-running activities with triggers and transactions. In: SIGMOD Conference, pp. 204–214 (1990)
22. Dayal, U., Hsu, M., Ladin, R.: A transactional model for long-running activities. In: Lohman, G.M., Sernadas, A., Camps, R. (eds.) VLDB Conference, pp. 113–122. Morgan Kaufmann, San Mateo (1991)
23. Dalal, S., Temel, S., Little, M., Potts, M., Webber, J.: Coordinating business transactions on the web. IEEE Internet Comput. **7**(1), 30–39 (2003)
24. Elmagarmid, A.K., Leu, Y., Litwin, W., Rusinkiewicz, M.: A multidatabase transaction model for interbase. In: McLeod, D., Sacks-Davis, R., Schek, H.-J. (eds.) VLDB Conference, pp. 507–518. Morgan Kaufmann, San Mateo (1990)
25. Elmagarmid, A.K. (ed.): Database Transaction Models for Advanced Applications. Morgan Kaufmann, San Mateo (1992)
26. Freund, T., Story, T.: Transactions in the world of web services, part 1. IBM DeveloperWorks, August 2002. http://www.ibm.com/developerworks/library/ws-wstx1/
27. Grefen, P.W.P.J., Apers, P.M.G.: Integrity control in relational database systems—an overview. Data Knowl. Eng. **10**, 187–223 (1993)
28. Grefen, P.W.P.J., Aberer, K., Ludwig, H., Hoffner, Y.: CrossFlow: cross-organizational workflow management for service outsourcing in dynamic virtual enterprises. IEEE Data Eng. Bull. **24**(1), 52–57 (2001)

29. Garcia-Molina, H., Salem, K.: Sagas. In: Dayal, U., Traiger, I.L. (eds.) SIGMOD Conference, pp. 249–259. ACM, New York (1987)
30. Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Mateo (1993)
31. Gray, J.: Notes on data base operating systems. In: Flynn, M.J., Gray, J., Jones, A.K., Lagally, K., Opderbeck, H., Popek, G.J., Randell, B., Saltzer, J.H., Wiehle, H.-R. (eds.) Operating Systems, an Advanced Course. Lecture Notes in Computer Science, vol. 60, pp. 393–481. Springer, Berlin (1978)
32. Gray, J.: The transaction concept: virtues and limitations (invited paper). In: VLDB Conference, pp. 144–154 (1981)
33. Grefen, P.W.P.J.: Advanced architectures for transactional workflows-or-advanced transactions in workflow architectures. In: International Process Technology Workshop (1999)
34. Grefen, P., Vonk, J.: A taxonomy of transactional workflow support. Int. J. Coop. Inf. Syst. **15**(1), 87–118 (2006)
35. Grefen, P.W.P.J., Vonk, J., Apers, P.M.G.: Global transaction support for workflow management systems: from formal specification to practical implementation. VLDB J. **10**(4), 316–333 (2001)
36. Grefen, P.W.P.J., Vonk, J., Boertjes, E., Apers, P.M.G.: Two-layer transaction management for workflow management applications. In: Database and Expert Systems Applications (DEXA), pp. 430–439 (1997)
37. Houston, I., Little, M.C., Robinson, I., Shrivastava, S.K., Wheater, S.M.: The CORBA activity service framework for supporting extended transactions. Softw. Pract. Exp. **33**(4), 351–373 (2003)
38. Härder, T., Reuter, A.: Principles of transaction-oriented database recovery. ACM Comput. Surv. **15**(4), 287–317 (1983)
39. Berkeley DB website (2007). http://www.sleepycat.com/products/bdb.html
40. IBM DB2 website (2007). http://www.ibm.com/db2
41. JSR-095 J2EE activity service for extended transactions 1.0 final release. java.sun.com (May 2006)
42. Jin, T., Goschnick, S.: Utilizing web services in an agent-based transaction model (abt). In: International Workshop on Web Services and Agent-Based Engineering (2003)
43. Jajodia, S., Kerschberg, L. (eds.): Advanced Transaction Models and Architectures. Kluwer, Dordrecht (1997)
44. Ku, K.-I., Kim, Y.-S.: Moflex transaction model for mobile heterogeneous multidatabase systems. In: Research Issues in Data Engineering (RIDE), pp. 39–46 (2000)
45. Kiepuszewski, B., Mühlberger, R., Orlowska, M.E.: Flowback: Providing backward recovery for workflow systems. In: Haas, L.M., Tiwary, A. (eds.) SIGMOD Conference, pp. 555–557. ACM, New York (1998)
46. Kratz, B.: Protocols for long running business transactions. Infolab technical report series, No. 17, Infolab, Tilburg University (2004)
47. Kratz, B.: A business aware transaction framework for service oriented environments. In: Hanemann, A. (ed.) Proceedings of IBM PhD Student Symposium at the 3rd International Conference on Service Oriented Computing (ICSOC). IBM Research Report RC23826, IBM Research Division (2005)
48. Kratz, B., Wang, T., Grefen, P.W.P.J., Vonk, J.: Flexible business transaction composition in service-oriented environments. Beta Technical Report WP 140, Eindhoven University of Technology (2005)
49. Lewis, P.M., Bernstein, A.J., Kifer, M.: Databases and Transaction Processing: an Application-Oriented Approach. Addison-Wesley, Reading (2001)
50. Little, M.: Business transaction protocol: transactions for a new age. Web Serv. J. **2**(11), 50–55 (2002)
51. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice Hall, New York (1999)
52. Little, M., Webber, J.: Introducing WS-CAF—more than just transactions. Web Serv. J. **3**(12), 52–55 (2003)
53. Moss, J.E.B.: Nested transactions: an approach to reliable distributed computing. PhD thesis, MIT, April 1981. Also available as Technical Report MIT/LCS/TR-260
54. Newcomer, E.: Context, coordinators, and transactions—the importance of WS-CAF, January 2004. http://www.webservices.org
55. Newcomer, E., Robinson, I., Feingold, M., Jeyaraman, R.: Web services coordination (WS-Coordination) version 1.1. OASIS, April 2007. http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.1-spec-os.pdf
56. Newcomer, E., Robinson, I., Freund, T., Little, M.: Web services business activity framework (WS-BusinessActivity) version 1.1. OASIS, April 2007. http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-os.pdf

57. Newcomer, E., Robinson, I., Little, M., Wilkinson, A.: Web services atomic transaction (WS-AtomicTransaction) version 1.1. OASIS, April 2007. http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-os.pdf
58. OMG—Object Management Group: Corbaservices: common object services specification. OMG document formal/97-07-04 (1997)
59. Oracle database website (2007). http://www.oracle.com/
60. Papazoglou, M.P.: Web Services: Principles and Technology. Pearson Education/Prentice Hall, New York (2007)
61. Pu, C., Kaiser, G.E., Hutchinson, N.C.: Split-transactions for open-ended activities. In: Bancilhon, F., DeWitt, D.J. (eds.) VLDB Conference, pp. 26–37. Morgan Kaufmann, San Mateo (1988)
62. Qi, Z., Xie, X., Zhang, B., You, J.: Integrating x/open dtp into grid services for grid transaction processing. In: International Workshop on Future Trends in Distributed Computing Systems (FTDCS), pp. 128–134. IEEE Computer Society, Los Alamitos (2004)
63. Ramamritham, K., Chrysanthis, P.K.: Advances in Concurrency Control and Transaction Processing. IEEE Computer Society, Los Alamitos (1997)
64. Reuter, A.: ConTracts: a means for extending control beyond transaction boundaries. In: Third International Workshop on High Performance Transaction Systems (1989)
65. Serrano-Alvarado, P., Roncancio, C., Adiba, M.E.: A survey of mobile transactions. Distrib. Parallel Databases **16**(2), 193–230 (2004)
66. Simon, E.: Distributed Information Systems: From Client/Server to Distributed Multimedia. McGraw-Hill, New York (1996)
67. Simple object access protocol (SOAP) version 1.2 part 0: Primer, June 2003. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/
68. Sheth, A.P., Rusinkiewicz, M.: On transactional workflows. IEEE Data Eng. Bull. **16**(2), 37–40 (1993)
69. Sumerians: language and writing. wikipedia.org (2007). http://en.wikipedia.org/wiki/Sumerians
70. Steinbach, T., Webber, J., Türker, C.: Proposed grid transaction rg-charter. http://www.data-grid.org/tm-rg-charter.html
71. Türker, C., Haller, K., Schuler, C., Schek, H.-J.: How can we support grid transactions? towards peer-to-peer transaction processing. In: Conference on Innovative Data Systems Research (CIDR), pp. 174–185 (2005)
72. Universal description, discovery and integration (UDDI) (2006). http://www.uddi.org/
73. Vonk, J., Grefen, P.W.P.J.: Cross-organizational transaction support for e-services in virtual enterprises. Distrib. Parallel Databases **14**(2), 137–172 (2003)
74. Vonk, J., Grefen, P.W.P.J., Boertjes, E., Apers, P.M.G.: Distributed global transaction support for workflow management applications. In: Bench-Capon, T.J.M., Soda, G., Min, A. (eds.) Database and Expert Systems Applications (DEXA). Lecture Notes in Computer Science, vol. 1677, pp. 942–951. Springer, Berlin (1999)
75. Vonk, J., Wang, T., Grefen, P.W.P.J.: A dual view to facilitate transactional qos. In: International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), pp. 381–382. IEEE Computer Society, Los Alamitos (2007)
76. Wang, T.: Towards a transaction framework for contract-driven, service-oriented business processes. In: Hanemann, A. (ed.) Proceedings of IBM PhD Student Symposium at the 3rd International Conference on Service Oriented Computing (ICSOC). IBM Research Report RC23826, IBM Research Division (2005)
77. Warne, J.: An extensible transaction framework: technical overview. Technical report, ANSA Architecture for Open Distributed Systems Project (1993)
78. Wang, T., Grefen, P.W.P.J., Vonk, J.: Abstract transaction construct: building a transaction framework for contract-driven, service-oriented business processes. In: Dan, A., Lamersdorf, W. (eds.) ICSOC Conference. Lecture Notes in Computer Science, vol. 4294, pp. 434–439. Springer, Berlin (2006)
79. Wächter, H., Reuter, A.: The contract model. In: Database Transaction Models for Advanced Applications, pp. 219–263. Morgan Kaufmann, San Mateo (1992)
80. Weikum, G., Schek, H.-J.: Concepts and applications of multilevel transactions and open nested transactions. In: Database Transaction Models for Advanced Applications, pp. 515–553. Morgan Kaufmann, San Mateo (1992)

81. Web services description language (WSDL) version 2.0 part 0: Primer, August 2005. http://www.w3.org/TR/2005/WD-wsdl20-primer-20050803/
82. Wang, T., Vonk, J., Grefen, P.W.P.J.: TxQoS: a contractual approach for transaction management. In: International EDOC Enterprise Computing Conference (EDOC), pp. 327–338. IEEE Computer Society, Los Alamitos (2007)
83. X/Open Company Ltd: Distributed transaction processing: reference model, version 3 (1996)