Workflow Transactions *

Johann Eder, Walter Liebhart
Department of Informatics, University of Klagenfurt, Austria
email: {eder, walter}@ifi.uni-klu.ac.at

January, 1996

Abstract

Process-oriented workflow systems require transactional support in order to guarantee consistent and reliable execution of business processes in a multi-user and non failure free environment. Classical (ACID) transactions are too constraining for long-lived computations, like workflow applications. Therefore, transaction relevant requirements of such applications are investigated and the basic concepts of a new kind of transactions - workflow transactions - are presented. The motivation is to integrate advanced transaction concepts into workflow models in order to define and support a transaction based execution of process-oriented workflows.

1 Introduction

Workflow management systems roughly may be divided into document-oriented and process-oriented systems [EL96a]. *Document-oriented* systems mainly support the flexible and often ad hoc coordination and cooperation of humans who are responsible for consistent execution results. *Process-oriented* systems control and coordinate the execution of (complex) business processes consisting of heterogeneous, distributed and/or autonomous tasks which are executed with little or no human intervention. Especially process-oriented systems require transactional support in order to ensure correct and reliable process execution in a multi-user and non failure free environment. Unfortunately, classical (ACID) transactions are in many aspects too constraining for workflow applications. Therefore numerous advanced transaction models have been developed, which allow a more complex transaction structure and/or relax (at least some of) the ACID properties. However, most of these newer transaction models offer valuable concepts for workflow applications but they are too database-centric and therefore not

*This research was supported, in part, by CSE-Systems, Computer & Software Engineering GmbH, Klagenfurt, Austria

directly applicable in the workflow context. *Workflow transactions* can be seen as an approach to fill the gap between advanced transaction models and process-oriented workflow models. How to realize workflows with workflow transactions is illustrated by the workflow activity model WAMO [EL95].

2 Basic Transaction Concepts

The basic idea of transaction processing is to guarantee a consistent and reliable execution of applications in the presence of concurrency and failures. A database transaction is a sequence of operations which transfers a database from one consistent state into another (not necessarily different) consistent state.Transaction processing technology ensures that each transaction either executed to completion or not at all, and that concurrently executed transactions behave as though each transaction executes in isolation. Additionally, these guarantees are upheld despite various types of failures (e.g. computer components). In general, these requirements are realized by a concurrency-control unit and a recovery unit.

Transaction processing systems pioneered many concepts in distributed and fault-tolerant computing. Most important, they introduced the transaction ACID properties - atomicity, consistency, isolation, and durability-that have emerged as the unifying concepts for distributed computing [GR93]:

- *Atomicity:* A state transition is atomic if it appears to jump from the initial state to the result state without any observable intermediate states - or if it appears as though it had never left the initial state (all-or-nothing principle).

- *Consistency:* A transaction produces consistent results only; otherwise it aborts. A result is consistent if the new state of the database fulfills all the consistency constraints of the application. Since it is impossible to check all constraints each time a transaction is started, it is assumed that the data is consistent in the initial database state or if it has been produced by a committed transaction.

- *Isolation:* Isolation means that a program running under transaction protection in a multi-user environment must behave exactly as it would in single-user mode. This topic is variously called consistency (the static property), concurrency control (the problem), serializability (the underlying theory), or locking (the technique).

- *Durability:* Durability requires that results of transactions having completed successfully must not forgotten by the system; from its perspective, they have become part of reality.

3 From Traditional Transactions to Advanced Transactions

In the database area *flat transactions* represent the most common (and simplest) type of transactions, and for almost all existing systems it is the only one that is supported at the application programming level. Nevertheless, they are too restrictive and inflexible for non-traditional applications, as for example workflows. These applications in general have the following characteristics:

- *Long duration:* Traditional transactions were invented for very short transactions whereas workflow activities have a much longer duration, touch many objects and have a complex control flow. Executing a long-running activity as a single ACID-transaction can significantly delay the execution of other high-priority short transactions, increase the probability of deadlocks (because of locking) and hence high transaction abort rates. In most cases it is necessary to externalize uncommitted results or make them visible to other activities in order to achieve acceptable performance. Of course, since the results are uncommitted they may become invalid later in the process. This fact in general can be tolerated from an application point of view but it requires adequate consistency preserving mechanisms (e.g. partial backward recovery with compensation). Additionally, it is not tolerable to rollback the whole workflow, and maybe the work of a day, if somewhere a failure (or exception) occurs.

- *Cooperation and concurrency*: In contrast to traditional applications, workflow activities are more of a cooperative nature where different subactivities are allowed to concurrently access shared, persistent data (e.g. working on a common document). Of course, there is some kind of synchronization necessary to control the concurrent access but workflows in general have much weaker synchronization requirements than traditional applications - for example, they tolerate inconsistent results to some extent. Serializability as a global correctness criterion is not applicable in the workflow domain because business processes themselves are *not serial.* Another important aspect are cooperation issues. There are not only intra-workflow dependencies (dependencies between activities within one workflow) but also *inter-workflow dependencies* (dependencies between different workflows) which must be supported adequately. Classical (ACID) transactions are seen as concurrent and completely unrelated units of work. This means that there are no *application independent* system services for specifying such dependencies except for putting all this control features into the application code.

- *Complex structure*: Flat transactions have only one layer of control which can be used by the application. Everything between `begin work` and `commit work` is at the same level which means that there is no way of committing or aborting parts of transactions, or committing results in several steps, and so forth.But especially transaction based workflow applications require more dimensions of control in order to manage the control flow over distributed and autonomous applications within a workflow.

The shortcomings of flat transactions motivated the development of more sophisticated - extended and relaxed - transaction models, as for example summarized in [Elm92]. *Extended transactions* permit grouping of their operations into hierarchical structures (e.g. nested transactions) and *relaxed transactions* indicate that (some of) the ACID requirements are relaxed (e.g. open nested transactions relax the isolation requirement) [RS94]. However, in defining new transaction models it must be kept in mind that the success of flat transactions was its *simplicity* in isolating the application from faults. Extending transactions for non-traditional applications is not a case of "the more the better"; rather, a delicate balance between expressive power and usability (simplicity) [GR93].

4 From Advanced Transactions to Workflow Transactions

As stated in [AKA$^+$95a, RS94] most of the advanced transaction models for non-traditional applications are developed from a database point of view, where preserving the consistency of the shared database by using transactions is the main objective. A basic fact behind these models is the attempt to use traditional transactions as building blocks which restricts the applicability in the workflow domain. Workflow activities are in general not database transactions which are started automatically and therefore the concepts of advanced transaction models cannot be applied directly.

Major work in expanding advanced transaction models for workflow requirements was done in the area of transactional workflows (e.g. [RS93, Hsu93, BDS$^+$93]) and long-running activities (e.g. [DHL91, WR92]). Nevertheless, this work still is mainly influenced by a database point of view and therefore not direct applicable for workflow systems.

Modern WFMSs have to support complex, long-running business processes in a heterogeneous and/or distributed environment. It has been pointed out in [GHS95] that most of these systems lack the ability to ensure correctness and reliability of workflow execution in the presence of failures. Therefore a strong motivation of merging advanced transaction models with workflow models becomes evident. *Workflow Transactions*

[EL96b] incorporate the basic ideas of traditional transactions extended by several features of advanced transaction models required for workflow execution. Currently there are several approaches reflecting this idea (e.g., [KS95, MAA$^+$95, AKA$^+$95b, Ley95, EL95]). The main characteristics of workflow transactions may be summarized as follows:

- *Analogies:* A flat transaction is a sequence of operations which transfers a database from one consistent state into another consistent state. A transaction is executed isolated from concurrent transactions and if one operation within a transaction fails, the whole transaction is rolled back. In analogy to this, a workflow transaction is a *sequence of workflow activities* which transfers a business process from one consistent state into the next consistent state. Activities themselves are again workflow transactions. From this point of view, we talk about workflow transaction on a more abstract level than we are used to talk about traditional transactions.

- *Transaction structure:* Workflow transactions must allow a hierarchical structure in order to be applicable for complex applications. A workflow typically consists of several activities which themselves again may be composed of (sub-) activities. Each activity is a workflow transactions. The nesting must be supported over as many levels as there are abstraction layers in the application. The most elementary activities finally represent an application program, a flat transaction or for example a human task (e.g. making a phone call).

- *Atomicity:* Since workflow activities are in general of long-duration, application dependent (user-defined) failure atomicity is required. The goal is not to undo everything in case of a failure but instead to *selectively roll back* parts of the work until the most recent *consistent* state (within the transaction) is reached. In order to find such a consistent state the workflow transaction manager needs support from a human expert (e.g. in WAMO [EL95] such states can be identified during runtime, based on specific information given by the workflow designer). Additionally, it would not be a realistic option in online systems to go back to the past (which corresponds to a rollback operation). Instead, more advanced recovery concepts [EL96b, Ley95] (especially compensation of already committed activities) have to be provided. Having reached such a point, forward execution of the process - perhaps on an alternative path - must be supported. Summing up, workflow transactions relax the "nothing" property of the all-or-nothing concept of traditional transactions.

- *Consistency:* As for traditional transactions, the scope of consistent executions does not focus on the work which is done within an activity (a transaction) but only on the correct execution order of activities. The commit of an activity is taken as a guarantee that the activity has produced a consistent result. If an activity aborts (fails) then an inconsistent state may be the consequence. As for flat transactions, such situations should not occur and must therefore be removed - ideally automatically. If compensating activities are involved in a recovery process then this activities have to terminate successfully in order to preserve consistency.

- *Isolation:* Because of the nature for workflow applications (long duration, cooperation, concurrency) it is not possible to execute workflow transactions fully isolated from concurrent transactions. As mentioned earlier, serializability as correctness criterion for concurrent processing is too restrictive. There exist several theoretical approaches to overcome this problem without compromising consistency (e.g. [BDS+93, Sch94], as for example semantic serializability. The goal is to exploit the semantics of the activities (by a human expert) by defining compatibility specifications between the activities. Compatibility between two activities means that the ordering of the two activities in the schedule is insignificant from an application point of view.

  Additionally, isolation is relaxed in the sense that subactivities externalize their results as soon as they commit in order to increase concurrency and hence performance. Of course, if the parent activity later on aborts or is involved in a compensation process then the results of the activity may have become invalid and therefore the activity has to be undone semantically (compensated). At this point it must also be emphasized that a compensation of activities is not always possible (e.g. drilling a hole cannot be undone later on).

- *Durability:* As soon as a workflow (sub-)transaction commits, its effects are persistent. Of course, these effects may be semantically undone later if the parent activity aborts or is compensated.

The workflow activity model WAMO [EL95] realizes most of the above presented concepts. WAMO supports the hierarchical structuring of workflows by using complex activities which are workflow transactions. There are several control flow operators which are defined over activities in order to specify the control flow between activities. Failure atomicity can be controlled by *vital* activities. Elementary activities (tasks) may be associated with compensation tasks which are necessary for backward recovery. Tasks which are not compensatable (at least without human intervention) are termed as *critical* tasks. WAMO's concepts currently are integrated into the prototype WFMS Panta Rhei [EGN94].

5 Conclusions

In order to ensure a consistent and reliable execution of process-oriented workflows, we have investigated workflow specific requirements for a transaction based execution of workflow applications. We started by discussing classical (ACID) transactions which are very successful in traditional applications (because of their simplicity) but too constraining for long-lived computations, as for example workflow applications. Then we focused on advanced transaction models, developed for non traditional applications. Unfortunately, these models are very database-centric and therefore not directly applicable in the workflow domain. Based on this fact we developed the concept of *workflow transactions* which allow a transaction based definition of workflows.

References

[AKA+95a] G. Alonso, M. Kamath, D. Agrawal, A. El Abbadi, R. Günthör, and C. Mohan. Advanced transaction models in workflow contexts. Technical report, IBM Almaden Research Center, 1995.

[AKA+95b] G. Alonso, M. Kamath, D. Agrawal, A. El Abbadi, R. Günthör, and C. Mohan. Failure handling in large scale workflow management systems. Technical report, IBM Almaden Research Center, 1995.

[BDS+93] Y. Breitbart, A. Deacon, H.-J. Schek, A. Shet, and G. Weikum. Merging application-centric and data-centric approaches to support transaction-oriented multi-system workflows. *SIGMOD RECORD*, 22(3):23–30, Sep. 1993.

[DHL91] U. Dayal, H. Hsu, and R. Ladin. A transactional model for long-running activities. In *Proc. of the 17th Int. Conference on VLDBs*, Barcelona, September 1991.

[EGN94] J. Eder, H. Groiss, and H. Nekvasil. A workflow system based on active databases. In G. Chroust and A. Benczur, editors, *CON 94: Workflow Management: Challenges, Paradigms and Products*, pages 249–265. Oldenbourg, Linz, Austria, 1994.

[EL95] J. Eder and W. Liebhart. The workflow activity model wamo. In *Proc. of the 3rd Int. Conference on Cooperative Information Systems*, Vienna, Austria, May 1995.

[EL96a] J. Eder and W. Liebhart. A workflow classification framework. Technical report, University of Klagenfurt, Department of Informatics, January 1996.

[EL96b] J. Eder and W. Liebhart. Workflow recovery. Submitted elsewhere, January 1996.

[Elm92] A.K. Elmagarmid. *Database Transaction Models for Advanced Applications.* Morgan Kaufmann Publishers, 1992.

[GHS95] D. Georgakopoulos, M. Hornick, and A. Shet. An overview of workflow management: From process modeling to workflow automation. In A. Elmagarmid, editor, *Distributed and Parallel Databases*, volume 3. Kluwer Academic Pub., Boston, 1995.

[GR93] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, 1993.

[Hsu93] M. Hsu. Special issue on workflow and extended transaction systems. *Bulletin of the Technical Committee on Data Engineering*, 16(2), June 1993.

[KS95] N. Krishnakumar and A. Shet. Managing heterogeneous multisystem tasks to support enterprise-wide operations. In A. Elmagarmid, editor, *Distributed and Parallel Databases*, volume 3, 1995.

[Ley95] F. Leymann. Supporting business transactions via partial backward recovery in workflow management systems. In G. Lausen, editor, *GI-Fachtagung: Datenbanksysteme in Büro, Technik und Wissenschaft*, Dresden, March 1995. Springer Verlag.

[MAA+95] C. Mohan, D. Agrawal, G. Alonso, A. El Abbadi, R. Günthör, and M. Kamath. Exotica: A project on advanced transaction management and workflow systems. *ACM SIGOIS Bulletin*, 16(1), 1995.

[RS93] M. Rusinkiewicz and A. Shet. On transactional workflows. *Bulletin of the Technical Committee on Data Engineering*, 16(2), 1993.

[RS94] M. Rusinkiewicz and A. Shet. Specification and execution of transactional workflows. In W. Kim, editor, *Modern Database Systems: The Object Model, Interoperability, and Beyond.* ACM Press, June 1994.

[Sch94] F. Schwenkreis. A formal approach to synchronize long-lived computations. *Proc. of the 5th Australasian Conference on Information Systems*, Melbourne 1994.

[WR92] H. Waechter and A. Reuter. The contract model. In A.K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 7. Morgan Kaufmann, 1992.