A Transaction-Oriented Workflow Activity Model

Johann Eder, Walter Liebhart
Institut für Informatik, Universität Klagenfurt
A-9020 Klagenfurt, Austria
email:{eder,walter}@ifi.uni-klu.ac.at

Abstract

The combination of workflow systems and database management systems significantly facilitates the design and reliable management of complex business processes. In this paper we present a high-level workflow activity description language and an advanced transaction mechanism to design long running activities. The strength of our model is the simplicity of the language and the application independent transaction facility which supports highly concurrent and reliable execution of workflow activities.

1 Introduction

Workflow technology is becoming more and more important for organizations to improve their productivity and competitive position through support, automation and reengineering of business processes (e.g. order processing, manufacturing, etc.). A *Workflow Management System* provides procedural automation of a business process by management of the sequence of work activities and the invocation of appropriate human and/or information technology (IT) resources. The *Workflow Management Reference Model* [1], defined by the Workflow Management Coalition Group, describes a common model, with three typical functional areas, for the construction of workflow systems:

- The *Build-time functions*, concerned with defining the workflow process and activities. They are necessary to transform a real world business process into a more formal and computer processable format. Such transformation methods are for example Petri Nets.
- The *Run-time control functions* concerned with interpreting and executing the modelled process description as well as coordinating and scheduling the execution of the various activities of each process. These functions are available through the *Workflow Enactment Service* (WF-Engine). Today there exist many types of workflow enactment environments like mail-based, database-centered or document-centered systems.
- The *Run-time interactions* with human users and IT application tools for processing the various activity steps. Individual activities within a workflow process are typically concerned with human operations and information processing operations requiring particular software systems (e.g., a mailer, an application program, a DBMS).

Most of the current available workflow systems offer interesting possibilities to manage complex business processes but in general they do not very well support automatic operation execution, monitoring the status of activities, enforcement of consistency and concurrency control, or recovery from failure. But especially these features are necessary to control and manage the concurrent execution of interdependent activities in complex business processes.

In this paper we want to focus on *Transactional Workflows* which can be seen as a combination of workflow systems and database management systems (DBMSs) with the intention to incorporate the advantages of both technologies [2]. One of the main goals is to provide mechanisms for defining and controlling long-lived activities, complex and concurrent computations, just like transactions in traditional DBMSs control short computations.

During the last years several new interesting transactional workflow models and concepts [3, 4, 5] (see also subsection 2.1) have been introduced and some basic features of these models have also been integrated into our model. However, the main difference to this related work is that our model is much easier to work with as it does not require skilled programmers and it is more flexible because of the possibility to use selectively easy but expressive control structures and intuitive simple transactional characteristics during the specification of business processes. Additionally, our model does not assume the existence of a database system with an advanced transaction mechanism (e.g. nested transactions) although this could improve the capabilities of our model.

The remainder of this paper is organized as follows: In section 2 we illustrate the necessity of *advanced transaction concepts*, especially for reliable workflow activity management. In section 3 we present a *high level description language* for the static and the dynamic description of workflows with an integrated and *application independent* advanced transaction model. A small example visualizes our concepts. Section 4 concludes this paper and presents a short overview of future work.

2 Advanced Transaction Concepts for Workflow Activity Management

2.1 The Evolution of Transaction- and Workflow Models

The traditional transaction model is defined by the *ACID properties* (Atomicity, Consistency, Isolation and Durability) and the notion of (conflict) serializability as correctness criterion for schedules (histories). Advanced transaction models can be classified according to various characteristics including transaction structure, intra-transaction concurrency, execution dependencies, visibility, durability, isolation, failure atomicity, and correctness criteria for concurrent execution[4]. *Extended transactions* permit grouping of their operations into hierarchical structures and *relaxed transactions* indicate that a given transaction model relaxes (some of) the ACID properties and the rigid serializability constraint.

A first important step in the evolution of traditional (flat) transactions was the development of *Nested Transactions* [6]. A nested transaction may recursively contain any number of subtransactions, forming a transaction tree. The root of this tree is called top-level transaction which satisfies the ACID properties. The main advantages of the nested transaction model are the support of *modularity* (decomposition of transactions), *failure handling* at the granularity of subtransactions and *intra-transaction parallelism* because subtransactions can be executed concurrently. *Open Nested Transactions* [7] relax the isolation requirements of nested transactions and make the results of committed subtransactions visible to other concurrently executing nested transaction. The concept of *compensation* opened a new dimension in the evolution of transaction models. If a subtransaction is compensatable then it can commit and release the resources before the parent transaction successfully completes and commits. If the parent transaction later aborts, the effects of the already committed subtransactions can be (semantically) undone by executing the corresponding compensation transactions. *Saga* [8] is an example of a transaction model with this feature. A lot of work is also done in the area of distributed transaction management as for example the development of the *DOM Transaction Model* [9] which allows closed nested and/or open nested transactions. The *Flexible Transactions Model* [10] is based on the nested transaction model and has been proposed as a transaction model suitable for a multi-database environment.

Important issues related to transactional workflow models besides [8, 9, 10], for example, were addressed in the work of *Long-Running Activities* [3], *On Transactional Workflows* [4] or *The ConTract Model* [5]. In [3] a *Long-Running Activity* consists recursively of multiple application steps each of which is either an activity or a (nested) transaction. Control flow and data flow of an activity may be specified statically in the activity's script or dynamically by ECA-rules. The model includes compensation, communication between steps and exception handling. In [4] a precise overview of *transactional workflows*, including a description of an abstract model of a task (by a state machine) is introduced. The *ConTract-model* [5] tries to provide the formal basis for defining and controlling long-lived, complex computations. *ConTracts* can be seen as a mechanism for grouping transactions into a multi-transaction activity. A *ConTract* consists of a set of predefined actions (with ACID properties) called steps, and a corresponding execution plan called a script.

A comprehensive common framework to specify and reason about the nature of interactions between transactions in a particular model is the *ACTA Metamodel* [11], which is also very helpful to "invent" new transaction models.

2.2 Transaction Requirements for Workflow Activities

One of the most fundamental drawbacks of traditional database transaction systems in the context of long-running activity management is the fact that transactions are seen as concurrent and completely *unrelated* units of work which means that there are no *application independent* system services for specifying intertransaction dependencies - like control flow or semantic connections - except for putting all this control features into the application code. Besides the former mentioned aspect there are a lot of other arguments why the ACID-properties and the rigid serializability correctness criterium of the traditional transaction model are too restrictive for activities and activity management in workflows, as for example:

- *Long-Lived-Activities:* Traditional transactions were invented for very short transactions but workflow activities normally have a 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.
- *More cooperation and concurrency through semantic serializability:* Workflow activities typically are more of a cooperative nature (e.g. working on a common document) where different subactivities concurrently access shared, persistent data and, therefore, some kind of synchronization is necessary. In this case, serializability as correctness criterion for concurrent access is too restrictive. One possibility to achieve acceptable performance without compromising consistency is to exploit the semantics of the application activities (by a human expert). Such *semantic serializability* [12] can, for example, be realized through *compatibility specifications* between activities. Compatibility between two activities means that the ordering of the two activities in the schedule is insignificant from an application point of view [7].
- *More cooperation and concurrency through relaxed isolation:* If subactivities (subtransactions) externalize *uncommitted* results or make them visible to other

subactivities then there is a higher level of concurrency and cooperation possible (in this case the isolation granularity is reduced to subactivities). But if the parent activity later on aborts it is necessary to *compensate* all committed activities which have used the uncommitted results. If compensation activities are available then the relaxation of the isolation property increases concurrency. But at this point it should be mentioned that compensation activities are not always available (eg. drilling a hole cannot be undone later on), such activities often are called *critical* activities, and that a compensation activity has to terminate successfully in any case to guarantee consistency.

- *Application dependent (user-defined) failure atomicity:* All-or-nothing in its original meaning is too restrictive and expensive for workflow activities. For example, it is not tolerable to rollback the whole workflow activity, and maybe the work of a day, if one, non very important (*non vital*) subactivity aborts. This can be avoided by introducing alternative (*contingency*) activities which are only executed if a non vital activity fails, by re-executing the activity or even by tolerating the failure so that the whole workflow can continue to make forward progress. Additionally, in case of a system failure a *forward recovery* of already committed activities has to be supported and compensatable activities are needed to handle the abortion of vital activities (*backward recovery*).

3 The Model

Our model enables the workflow designer to *easily* model complex business processes in a simple and straightforward manner. The basic idea is to decompose a complex business process into smaller work units (activities) which themselves consist of - ideally preexisting - tasks and to guarantee *reliable* flow control (including exception handling) by using control structures and other transaction specific constructs which are input to the transaction facility.

3.1 Activity and Task Specification

The basic constructs of our model are activities and tasks. Activities may *recursively* consist of other activities. Nested activities form an *activity tree*. The edges between activities represent different control structures and the leaves of the activity tree are tasks which are the elementary units of work (e.g. ACID-transactions, an application program or a human interaction). Every task and every activity has a *parent activity* except for the root activity. Tasks may not consist of other tasks or activities.

*Control structure*s are structuring mechanisms to support the decomposition of business processes into smaller work units and to define the flow of control between this work units in the activity tree. There are four simple but powerful control structures:

- *Sequence:* In a sequence activities are executed strictly sequential. This means that activity A_i cannot begin execution until activity A_{i-1} has terminated.
- *Ranked Choice:* This construct enables the modelling of alternative (contingency) activities which are activated when the previous activity terminates negatively.
- *Free Choice:* Similar to ranked choice but the activation order of the alternative activities in a free choice list is computed dynamically (at run time).
- *Parallel:* A parallel control structure enables activities and tasks to run concurrently.

3.2 Execution of Activities and Tasks

At run time activities are associated with unique identifiers and the modelled activity tree defines the execution order of activities and tasks in the *activity execution tree* (AET). Activities and tasks have different *execution states* during execution, as for example: active, not active, committed, aborted or compensated. Additionally, activities and tasks are able to react on events like: start, abort or commit. All activities and tasks in the AET are executed under the control of an advanced transaction manager. Main characteristics of the underlying transaction model are:

- *Relaxed Atomicity:* Each application may have its own application dependent failure atomicity (see section 2.2). A workflow may survive and make forward progress although some of its tasks do not terminate successfully.
- *Relaxed Serializability:* It is not possible to execute an entire workflow as a single isolated transaction to achieve (data) consistency. In our approach consistency is guaranteed by user defined *semantic serializability* between concurrent and interleaving workflows (inter-workflow dependencies) and correct execution of each individual workflow (intra-workflow dependencies). Nevertheless, the traditional serializability criterion will be necessary in many situations.
- *Relaxed Isolation:* Isolation will be relaxed if there are compensatable activities (see section 2.2). In this case activities may externalize uncommitted results and release resources to achieve a higher level of concurrency.

Many of the advanced transaction features are very easy to use and control by the workflow designer during construction of the AET, as for example:

- By the use of *control structures*: Control structures are simple but expressive mechanisms to handle task coordination requirements (intra-workflow dependencies).
- By the use of *transaction specific features*: Tasks can be specified more detailed by the STORNO-TYPE and FORCE parameter. Additionally, subactivities which are not essential for a successful termination of the parent activity can be defined by the NON VITAL parameter.

The STORNO-TYPE and FORCE parameter of a task are necessary for eventual compensation transactions. With the STORNO-TYPE the workflow designer may specify how a specific task behaves in case of compensation. There are four possibilities:

- *none:* The committed task does not need to be compensated because it is not relevant from an application point of view.
- *undoable:* The committed task can be undone by the corresponding compensation task without any side-effects. Let S be the database state at some time t, T the original task and TC the compensation task. Then the database state S' after executing T and TC in sequence equals the previous state S if in the between time no other operation was executed. (e.g. a client makes a flight reservation - later he cancels the reservation without paying a cancellation fee).
- *compensatable:* The committed task can be *semantically* undone by the corresponding compensation task but there are side-effects. Let again S, S' be database states, T the original task and TC the compensation task. Then the database state S' after executing T and TC in sequence may not be equal to the previous state S regardless whether in the between time other operations have been executed or not. (e.g. a client makes a

flight reservation - later he cancels the reservation but now he has to pay a cancellation fee).

- *critical:* The task cannot be undone or compensated in case of failure because there exists no compensation task to undo the already committed effects (e.g. drilling a hole, mailing a sensitive information, etc.). As we want to be compatible to existing database transaction systems we do not assume a *visible two-phase* commitment protocol which would simplify the handling of critical components.

Some tasks in real world situations are always expected to terminate successfully. This natural feature may also be demanded from tasks in our model by using the parameter FORCE during the specification of a task (e.g. open an account should always be possible). Such tasks are repeated and re-executed in case of failure (e.g. deadlock, unavailable resource, etc.) until a positive acknowledgement is achieved.

There also may exist activities within a workflow AET which are *not essential* for the parent activity to terminate successfully. For such parent-child relations we have introduced the transaction specific parameter NON VITAL. If a non vital activity fails the workflow can continue and make forward progress without any compensation actions. Normally, all (sub) activities within a workflow AET are essential for the parent activity to terminate successfully. If a *vital* activity fails then the compensation mechanism is activated. For example: if a vital activity in a sequence fails then the whole sequence fails which means that all previous committed activities and tasks in the sequence have to be compensated.

The workflow coordination requirements (control- and data flow) between work units can be described *formally* by *dependency rules* based on valid state transitions between nodes in the AET. The possible state transitions mainly depend on states and output values of other activities or tasks. We have formally defined these dependencies on basis of the ACTA Transaction Metamodel [11] in [13] and we will implement our model by the use of rules in an active database system.

3.3 Definition of the Workflow Activity Description Language

For the formalization of a complex business process we have developed a simple to use high-level **W**orkflow **A**ctivity **D**escription **L**anguage (WADL). As already explained, the basic constructs of our language are activities, tasks, control structures and transaction specific parameters. We now introduce WADL with the following syntactic sketch:

```
DEFINITION ACTIVITY A_ID
        SEQUENCE          [non vital]  A  {[non vital] A}          END-SEQUENCE          OR
        RANKED CHOICE     A {A}                                     END-RANKED-CHOICE     OR
        FREE CHOICE       A {A}                                     END-FREE-CHOICE       OR
        PARALLEL          [non vital] A [non vital] A {[non vital] A}   END-PARALLEL      OR
        TASK
END-ACTIVITY-DEFINITION

DEFINITION TASK [STORNO-TYPE] [FORCE]
        ACID-Transaction | ApplicationProgram | HumanInteraction
        INVERSE_TASK      % INVERSE_TASK is  necessary if STORNO-TYPE = ( COMPENSATABLE or UNDOABLE)
END-TASK-DEFINITION
```

3.4 A Small Example

In this subsection we present a simplified workflow example which emphasizes the most important features of our model. The example is illustrated graphically and also more formal by the Workflow Activity Description language:

```
ACTIVITY Trip_Reservation
        SEQUENCE
                        Flight_Reservation
          NON VITAL Car_Room_Reservation
                        Payment
        END SEQUENCE
        ACTIVITY Flight_Reservation
                SEQUENCE
                                F_Res
                  NON VITAL Notification
                END SEQUENCE
        END ACTIVITY Flight_Reservation
        ACTIVITY F_Res
                TASK F_Res
        END ACTIVITY F_Res
        ACTIVITY Car_Room_Reservation
                PARALLEL
                                Room_Res
                  NON VITAL Car_Res
                END PARALLEL
        END ACTIVITY Car_Room_Reservation
        ACTIVITY Payment
                FREE CHOICE
                        Cash
                        Credit_Card
                        Cheque
                END FREE CHOICE
        END ACTIVITY Payment
END ACTIVITY Trip_Reservation
TASK F_Res (COMPENSATABLE)
        F_Res
        INVERSE_TASK Comp_F_Res
END TASK F_Res
...
```

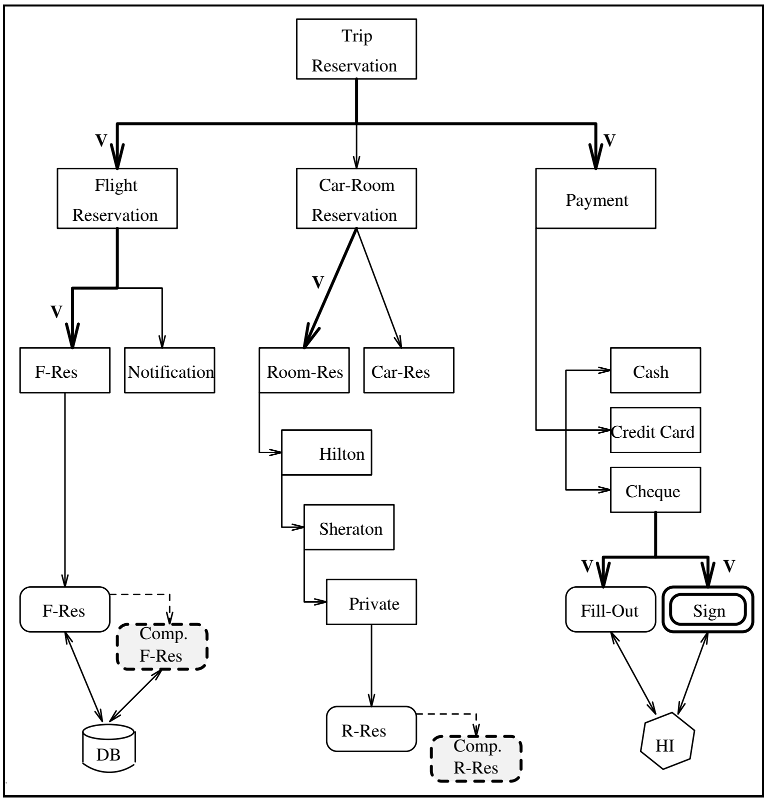

Figure 1: Trip Reservation

A *Trip Reservation* consists of the activity sequence *Flight Reservation* - *Car-Room Reservation* and *Payment*. *Car-Room Res.* is not vital for the parent-activity *Trip Res.*, whereas *Flight Res.* and *Payment* are vital activities. This means if, for example, *Payment* terminates negatively then *Car-Room Res.* and *Flight Res.* have to be undone (compensated). The *Car-Room Res.* consists of two parallel executable activities: *Room-Res* and *Car-Res*. The vital *Room-Res* is modelled as a ranked choice (if there is no room at Hilton then a reservation at Sheraton is tried, etc.). If the non vital *Car-Res* fails, *Car-Room Reservation* still may terminate successfully. The *Payment* is modelled as a free choice which means that at execution time it is decided which kind of payment is wished. If the *Payment* is done by *cheque* then the two tasks *Fill-Out* and *Sign* are activated which are vital human interactions (HI). In addition, *Sign* is a *critical* task and cannot be undone later. The two tasks *F-Res* and *R-Res* can be compensated by the corresponding tasks *Comp-F-Res* and *Comp-R-Res*.

4 Conclusion

Transactional workflows can be seen as a combination of workflow management systems and database management systems with the aim to produce *reliable* control flow management for long running workflow activities.

Our model allows the design of complex business processes by combining (preexisting) tasks and activities which will be executed and coordinated under the control of an advanced transaction management facility. In particular, we have presented a high-level *Workflow Activity Description Language (WADL)* to model such transactional workflows. The strength of the language mainly is based on simple structuring mechanisms to design complex business processes and the possibility to express application specific transactional requirements. Typically, workflow activities are of a long duration, highly concurrent and of a cooperative nature. Therefore, our transaction facility supports the relaxation of atomicity, serializability and isolation as well as, for example, the possibility to compensate activities.

Currently we are defining formally the static safety of activities and tasks in the activity tree and additionally, we are evaluating techniques for increasing concurrency and co-operation. The proposed transaction-oriented workflow activity model will be integrated in our prototype system based on an *active* database management system [14]. We will also develop a graphical application development interface to design such workflows with the possibility to generate WADL-code automatically.

References

[1] The Workflow Reference Model. Workflow Management Coalition Specification Document, Version 0.6, June 1993.

[2] McCarthy D.R., Sarin S.K.: Workflow and Transactions in InConcert. Bulletin of the Technical Committee on Data Engineering, Vol. 16, No. 2, June 1993.

[3] Dayal U., Hsu H., Ladin R.: A Transactional Model for Long-Running Activities. Proc. of the 17th Int. Conf. on VLDBs, Barcelona, Sept. 1991.

[4] Rusinkiewicz M., Shet A.: On Transactional Workflows. Bulletin of the Technical Committee on Data Engineering, Vol. 16, No. 2, 1993.

[5] Wächter H., Reuter A.: The Contract Model. In [15].

[6] Moss J.E.B.: Nested Transactions: An Approach to Reliable Distributed Computing. MIT Press, Cambridge, MA, 1985.

[7] Weikum G., Scheck H.-J.: Concepts and Applications of Multilevel Transactions and Open Nested Transactions. In: [15].

[8] Garcia-Molina H., Salem K.: SAGAS. Proc. of ACM SIGMOD Conf. on Management of Data, 1987.

[9] Buchmann A., Özsu M.T., et al.: A Transaction Model for Active Distributed Object Systems. In: [15].

[10] Elmagarmid A.K., Leu Y., Litwin W. Rusinkiewicz M.: A Multidatabase Transaction Model for InterBase. Proc. of the 16th VLDB Conf. Brisbane, Australia 1990.

[11] Chrysanthis P. K., Ramamritham K.: ACTA: The Saga Continues. In: [15].

[12] Breitbart Y., Deacon A., et al.: Merging Application-centric and Data-centric Approaches to Support Transaction-oriented Multi-system Workflows. SIGMOD RECORD, Vol. 22, No. 3, Sept. 1993.

[13] Eder J., Liebhart W.: A Formal Description of the Workflow Activity Description Language WADL. Technical Report, University of Klagenfurt, Austria, 1994.

[14] Eder J., Groiss H., Nekvasil H.: A Workflow System Based on Active Databases. 9th Austrian-Hungarian Informatics Conference, Austria, Oct. 1994.

[15] Elmagarmid A. K.: Database Transaction Models for Advanced Applications. Morgan Kaufmann, 1992.