Workflow Recovery *

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

Abstract

Workflow management systems (WFMSs) more and more become the basic technology for organizations to perform their daily business processes (workflows). A consistent and reliable execution of such processes is crucial for all organizations. We claim that this can only be achieved by integrating transactional features - especially "workflow transactions" - into WFMSs. Based on this idea, we discuss in detail advanced workflow recovery concepts which are necessary for the reliable and consistent execution of business processes in the presence of failures and exceptions. Additionally, we distinguish between different workflow types and present adequate recovery concepts for each of them.

1. Introduction

Business processes typically consist of multiple activities which have to be performed in a valid sequence. Activities themselves, represent any unit of work (e.g., a specific application program, a phone call) which may be characterized as heterogeneous, autonomous and / or distributed. Workflow management systems (WFMSs) are expected to control the execution of such business processes. In this sense, a WFMS represents a cooperative information system which handles the co-ordination and co-operation between the activities, constituting a business process. Within the process each activity has a more or less strong influence on the overall success of the process. Severe problems may arise, if failures occur (e.g. an activity does not behave in the expected manner). WFMSs should be able to react flexible on failures and they should ensure a correct and reliable execution of processes in the presence of concurrency and failures. But as has been pointed out in [11], almost none of the current commercial WFMSs support such a functionality. Similar problems have been discussed in the database area where the idea of transactions has solved many of the problems. Unfortunately, these concepts are not directly applicable in the workflow domain but they offer at least valuable ideas and concepts which are relevant for WFMSs. To guarantee reliable and consistent workflow executions the introduction of some kind of transactions in WFMSs is unavoidable. Because of the differences to traditional database transactions and also to most of the advanced transaction models we call them *workflow transactions*. In contrast to advanced transaction models, workflow transactions focus on issues of consistency from the business point of view rather than from a database point of view. A motivating example for the introduction of workflow transactions into WFMSs is illustrated in figure 1.

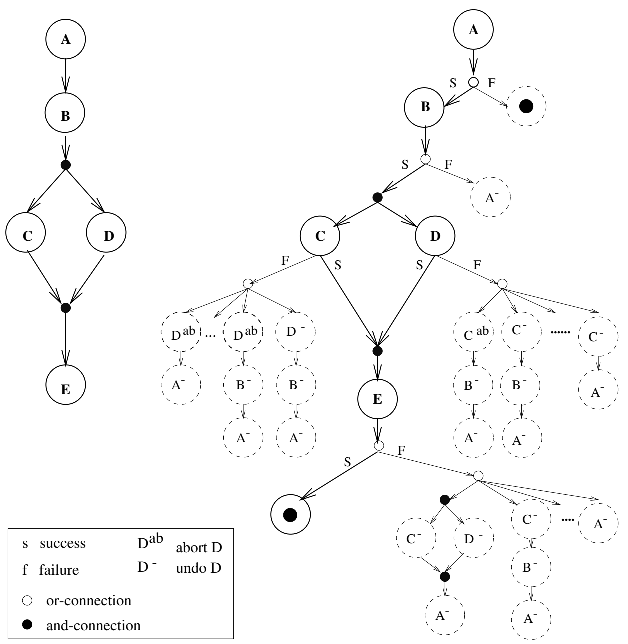

Figure 1. Motivation for Workflow Transactions

*This research was supported by the project "Workflow Transactions" sponsored by CSE-Systems, Computer & Software Engineering GmbH, Klagenfurt, Austria

This example stresses, how complex a rather simple workflow can become by trying to handle possible failures explicitly in the process description. The workflow on the right side within the example represents the simple workflow on the left side extended with possible failure paths. Of course, it is neither possible nor intended by most workflow designers to model *all* failures but the process description will in any case become complex very soon - especially if the original process is more complex. By introducing workflow transactions into WFMSs this problem can be minimized considerably because workflow transactions support the automation of failure handling during runtime (this means that the right side in figure 1 is computed automatically during runtime).

Within this paper we present a detailed overview of workflow recovery concepts which are necessary within a transaction based workflow execution. The necessity and advantages of workflow transactions has already been discussed in [8] by introducing the workflow model WAMO. Based on this model we point out relevant recovery concepts and we also introduce recovery concepts which go beyond the model. In section 2 the *Workflow Activity MOdel WAMO* is briefly discussed and related work is presented. Section 3 points out specific workflow characteristics which influence workflow recovery. In section 4 we discuss workflow recovery concepts in detail. Section 5 describes possible realization concepts for workflow recovery and section 6 concludes the paper.

2. Current Approaches

In this section we present a short overview of WAMO since the model is used to illustrate recovery concepts. Additionally, we discuss related work where we mention relevant approaches in the area of transaction based workflows.

2.1. The Workflow Activity Model WAMO

The workflow activity model WAMO [8] 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. Additionally, simple control structures enriched by transactional features allow the definition of failure-tolerant processes.

WAMO's metamodel not only incorporates traditional workflow modeling features but also transactional features. A workflow typically consists of multiple *activities*, *data objects* and *agents*. Activities represent any abstract description of work units in the business process. Data objects (e.g. complex forms) represent the information which is exchanged between activities. An agent is either a human or any computer system which is responsible for the execution of an activity.

WAMO supports hierarchical structuring of workflows by using *complex activities* which consist of other (sub-) activities, representing *subprocesses*. Furthermore, a certain activity may take part in several other activities, especially several times in an other activity which enhances re-usability. New workflows can easily be composed by re-using activities. Additionally, complex activities support the modeling of control structures (behavior) over activities. Up to now, WAMO offers the following simple but powerful control structures: *sequence*, *parallel*, *nesting*, *ranked choice* and *free choice*. The choice constructs enable the modeling of alternative (contingency) activities. An alternative activity is executed only if the immediate previous activity fails. In contrast to a ranked choice, the execution order in a free choice list is computed dynamically (at run time). Elementary activities - called *tasks* - are activities which are not further decomposable. From the workflow designers point of view, they are black boxes which finally perform the real work within a workflow. For the rest of our discussion we do only distinguish between tasks and activities if it is really necessary. If we use the term activity, it should be clear from the context whether a complex activity is meant or an elementary task or both.

At run time activities are associated with unique identifiers and the previous described control structures define the execution order of the activities. Activities and tasks have different execution states during execution (e.g. failed, compensated) and, additionally, they are able to react on specific events (e.g start, compensate). The correct execution order of activities is fully under the control of an advanced transaction manager. The underlying advanced transactional features are very easy to handle by the workflow designer during process specification, as for example:

- *by control structures:* Control structures are simple but expressive mechanisms to handle activity coordination requirements (intra-workflow dependencies).

- *by transaction specific features:* Tasks can be specified more detailed by the storno type and force parameter. Additionally, activities which are not essential for a successful termination of the corresponding parent activity can be defined as non vital.

  WAMO's transactional features focus primarily on the consistency of business processes in contrast to similar features within advanced transaction models, as for example in [23, 5], which focus on the consistency of (multi-) database systems.

The storno type and force parameters of a task are necessary for workflow recovery (i.e., for compensation). With the storno type the workflow designer specifies how a specific task behaves in case of a compensation. There are four possibilities: Type none (1) means, that the committed task does not need to be compensated because it is not necessary from an application point of view. Type undoable (2) means that a committed task can be undone by a corresponding compensation task without any side-effects, in the sense of an inverse operation. Type compensatable (3) means that the compensation of a task leads to some side-effects (e.g. money transfer and back transfer with transfer fees). Type critical (4) means that a task which has already terminated cannot be undone or compensated afterwards because its effects are irreversible within the current context (e.g., drilling a hole, mailing a sensitive information).

Some tasks in real world situations are always expected to terminate successfully (e.g., open an account, print a document) - although it may take several attempts. Therefore WAMO offers the task specific parameter force. Forcable tasks simplify recovery actions in case of a failure because they can be repeated and re-executed several times (specified by the workflow designer) until a positive acknowledgment is achieved - otherwise, process execution stops for manual intervention.

Another important transaction relevant feature is the concept of vital and non vital activities in order to specify the importance of a specific subactivity for its parent activity. If a non vital activity fails, the workflow can continue and make forward progress without any compensation actions. Normally, all activities within a workflow are essential and therefore vital for the parent activity. In any case, if a vital activity fails then the compensation mechanism is activated (this concept is explained in detail in section 4).

Most of the previous mentioned concepts are currently integrated into our prototype WFMS *Panta Rhei* [7].

2.2. Related Work

The integration of transactions into workflows was motivated by research efforts concerning database transaction models for advanced applications, as for example summarized in [10]. As stated in [2] most of these models 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 and the focus on specific applications with sometimes rather restrictive transactional features (e.g. rigid compensation policies in [26] which restrict the applicability in the workflow domain. Therefore the concepts of advanced transaction concepts cannot be applied directly.

Major work in expanding advanced transaction models for workflow requirements was done in the area of transactional workflows [24, 13, 4]. Nevertheless, this work still is mainly influenced by a database point of view which leads to rather restrictive models.

Modern WFMSs have to support complex, long-running business processes in a heterogeneous and/or distributed environment. As has been pointed out in [11] most of these systems lack the ability to ensure correctness and reliability of the workflow execution in the presence of failures. However, currently there are several approaches to overcome this shortcoming. In the METEOR project [18] a computational model for workflows is presented which captures the behavior of both, transactional and non-transactional tasks of different type. Additionally, two languages have been designed to address the important issues of inter-task dependencies, data formatting, data exchange, error handling, and recovery. Another example is IBM's Exotica project [22, 3] which aims at exploring several research areas from advanced transaction management concepts to client/server architectures and mobile computing within the context of workflow management. The goal is to incorporate at least some of the results into the commercial WFMS IBM Flowmark[1].

Concerning *workflow recovery* there are only a few research activities to name. A first discussion was presented in [15]. Additionally, the necessity of workflow recovery concepts is slightly addressed in [11] and [16]. More specific work in this area is presented in [19, 3]. Especially, the concept of business transactions, introduced in [19], describes some basic workflow recovery ideas in detail (above all partial backward recovery). Nevertheless, there exists no broad discussion about workflow recovery and this paper may be seen as a first deeper step into this important workflow area.

3. Workflow Characteristics influencing Workflow Recovery

In order to discuss workflow recovery concepts in detail it is first necessary to analyze the areas which influence the recovery process. First, we believe that it is not possible to present a general solution for workflow recovery because there exist different workflow types which demand different recovery approaches. Therefore, we present very briefly a workflow classification architecture which helps us to identify possible workflow types. Second, recovery actions are only necessary because of the existence of failures and exceptions. Therefore, it is absolutely necessary to analyze at least the most important failure types which may occur within workflow execution.

[1]Flowmark is a trademark of International Business Machines Corp.

3.1. Workflow Classification

Up to now there exists no general accepted classification framework for workflows (processes) and workflow systems. Since every classifications focuses on some specific aspects, it will always be difficult and probably impossible to give a commonly accepted classification. In [11] workflows are characterized along a continuum from *human-oriented* to *system-oriented workflows*. In the first case, a workflow is mainly performed by human agents. The WFMS is expected to support the coordination and collaboration of humans who are responsible for consistent workflow results. In the second case, workflows are characterized as highly automated and computation-intensive processes which involve the integration of heterogeneous, autonomous and / or distributed information systems. Since human influence is very limited, system-oriented workflows must include software for various concurrency control and recovery techniques to ensure consistency and reliability.

Another classification distinguishes between ad hoc, administrative, and production workflow [21]. The main differences between these types comprise (1) repetitiveness and predictability of workflows and tasks, (2) how the workflow is controlled (e.g., human controlled or automated) and (3) requirements for WFMS functionality. According to the previous classification *ad-hoc* and *administrative workflows* are closer to the human-oriented end of the spectrum whereas *production workflows* (e.g. trip reservation, loan requests, insurance claims or telecommunication processes) represent more complex business processes which communicate with different information systems and are hence closer to the system-oriented end of the continuum.

As we will see at the end of this subsection, our classification approach [9] leads to similar results but it points out some specific features which are later relevant within the recovery concepts. The classification is based on two pillars: the static and the dynamic aspects of a workflow.

3.1.1 Static Aspects of a Workflow

The static aspects of a workflow comprise all components which can be extracted from a workflow metamodel. According to WAMO's metamodel[8] a workflow essentially consists of *activities*, *data objects* and *agents* (refer to section 2.1.). We will now look more detailed at these components:

- *Activities*: Activities - especially tasks - describe the real work items in a process. In a first step, we have to distinguish between manual and automatic (either interactive or non interactive) tasks. *Manual tasks* are performed mainly by human agents. This includes in particular the manual start and manual termination of tasks. The work which is performed within a manual task is more or less fully under the control of the human agent (e.g. making a phone call, writing a letter). Hence, the WFMSs support in performing this work is rather limited and in general reduced to provide the agent with appropriate standard tools (like a text processor). On the other side, *automatic tasks* can be much better supported by a WFMS in the sense that the system can start automatically a task, compute the next step in the process and so on. *Interactive automatic tasks* are associated with *specific* programs (software applications) which are executed after a responsible human agent has selected the task from his worklist. During task execution the agent communicates interactively with the associated program. As soon as the program terminates also the task reaches its termination state. *Non interactive automatic tasks* are specific batch programs which are fully executed under the control of the WFMS - the program is started automatically by the WFMS and executed completely without human interaction.

- *Data objects*: Within a workflow different kinds of data - primarily depending on how much the WFMS knows about the semantics of the data - are relevant. We have to distinguish between data which is manipulated within tasks and data which is needed for process execution (scheduling, etc). Of course, these different kinds of data are not necessarily disjunctive (e.g., the amount of a loan in a loan request workflow is used within tasks but also for workflow scheduling in order to find the correct path in the process). More precisely, in a workflow application we find the following data types:

  - *Application (case) data:* The application data is consumed and produced by the various tasks (applications) in a workflow. It is further reasonable to classify application data into *structured*, *unstructured* or *semi-structured* data in order to investigate how easy this type of data can be used for process specification. Based on this classification it is easy to conclude that structured data (e.g. formatted data in a form) is not only suitable for application specific usage but also for process definitions.

  - *Process data:* Process data are necessary to define and control the execution of workflows. Typical examples are the state of tasks, the start time of a task and so on.

- *Agents*: An important function of a WFMS is to assign tasks to agents who are eligible to carry them out. The modeling and definition of agents composes very simple but also very sophisticated approaches [6]. Within our classification we distinguish only between human and machine agents.

3.1.2 Dynamic Aspects of a Workflow

The execution of a workflow mainly comprises the answer to the following question: *What (activity) has to be executed when, by whom and with which data?* In [14] these W-questions are termed as functional (what), behavioral (when), organizational (whom) and informational (which data) perspectives. We want to emphasize that the temporal execution order of the various activities within a workflow is a central topic. Based on this perception, we informally define a workflow as *the definition and/or execution of correct activity-sequences.* Such sequences can be defined as follows:

- *Ad hoc and without a corresponding formalism:* Within this approach correct activity sequences are determined by human agents ad hoc during workflow execution (at run time). Additionally, we distinguish whether the correct sequences are defined with *predefined* activities or not. In the first case, a workflow is composed of already defined parts at run time while in the other case the agents have the possibility to define *new activities* (and hence workflows) during run-time.

  This concept is also valid for WFMSs which are not based on activities but on agents (e.g. email based WFMSs). In this terminology each agent has to decide who should be triggered next in order to perform some work (which corresponds to an activity).

- *With a corresponding formalism:* Valid activity sequences are defined during workflow modeling time (at build-time) by the workflow designer. Therefore most WFMSs offer corresponding modeling tools which allow the definition of business processes based on a special formalism. Different formalisms are discussed in [9]. Main differences between the various formalisms depend (1) on the available process information and (2) on the access to (structured) application data.

3.1.3 Workflow Types

By investigating several workflow systems (and the processes they support) according to our classification, we identified at least two main types of workflows which we call document-oriented and process-oriented workflows.

- *Document-oriented workflows* are primarily composed of *manual* tasks and *unstructured or semi-structured* data elements (especially documents). Task ordering and coordination is either predefined roughly at build-time or on the fly during process execution. Hence, the execution of document-oriented processes is mainly controlled by the workflow participants. They decide *when* a specific task is performed, *which* task should be executed next (in case there is a set of potential successor tasks) and *who* should execute the next task. The requirements for WFMSs in this area are to support the coordination and collaboration of humans who are responsible for consistent execution results.

- *Process-oriented workflows* are much more populated by *automatic* tasks which manipulate above all *structured or semi-structured* data objects. As the processes tend to be very complex, adequate formalisms are necessary to define them. WFMSs which handle such processes are expected to control and coordinate the execution of the tasks with little or no human intervention. Therefore, various features concerning robustness, concurrency and recovery are necessary to guarantee a reliable and consistent execution. Of course, these requirements also restrict the user flexibility in executing workflows.

3.2. Failure and Exceptions

Many practical experiences have proven that the handling of failures and exceptions requires a lot of time and costs. Because of this reason, the adequate treatment of this problematic is a key success factor for cooperative information systems, especially for WFMSs. Since there are so many different types of failures we first identify potential *failure sources* and after that we give a summarizing classification of various failure classes.

Potential failure sources in a workflow comprise (1) workflow engine failures, (2) activity failures and (3) communication failures (see figure 2).

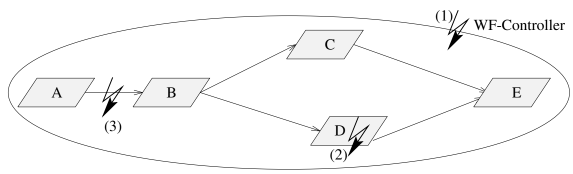

Figure 2. Basic failure sources

1. *Workflow engine (controller) failures*, like a system breakdown, lead to an abnormal termination of workflow execution. The goal of recovery now is to restore the "latest consistent" state of the system in order to resume process execution. This may be exactly

the state at the time the failure occurred or the latest consistent state immediately before the failure but also a new consistent state after the failure. The first two cases can be handled by some kind of *crash recovery*. This means for example, that all worklists are brought into the same state as they had been at the time the failure occurred or into a previous consistent state (see also [20]). The third case means that activities which had been *active* somewhere on a client at the time the failure occurred, may have terminated in the meantime. Such activities should not be aborted in case of a workflow engine failure. Instead crash recovery must be extended by some kind of *forward recovery* which brings the system into the new state. After updating the worklists, process execution can be continued. To reduce the amount of work in case of recovery after a workflow engine failure it is advisable to keep process relevant information within a database management system in order to use the recovery facility of the database. The prototype workflow system Panta Rhei [7] is build on top of a DBMS which simplifies crash recovery treatment. The problem of distributed activities which are executed autonomously during a workflow engine failure is more precisely discussed in [1].

2. *Activity failures* comprise failures within an activity. Activity failures are the primary subject of this paper and are therefore discussed more precisely in the rest of this paper.

3. *Communication failures between the scheduler and activities* are the source of another type of failure within workflow execution. The coordinated execution of distributed activities demands a stable communication between scheduler and activities. If, for example, the scheduler starts an activity, then the scheduler wants to be informed about the result of this operation. Middleware-components, like TP-monitors [12] support the handling of such requirements.

As stated before, within this paper we concentrate our discussion primarily on activity failures and to some extend on communication failures. For that purpose, we will now present very briefly the results of a failure classification we discussed in [8]. There, two main classes of failures concerning activities have been identified: System failures and semantic failures.

- *System failures* comprise information technology and application failures which lead to an abnormal termination of an activity (e.g. system breakdown, division by zero, deadlocks) - *the activity aborts*.

- *Semantic failures* comprise exceptions *within* the business process (i.e. they are not caused by information technology) which lead to a negative (but not abnormal) termination of the activity (e.g., the activity hotel reservation fails, because the hotel is already over-booked, the hotel reservation is interrupted by the user). If an activity is involved in a semantic failure then *the activity fails*.

4. Concepts for Workflow Recovery

A business process typically consists of multiple activities and each of these activities has a more or less strong influence on the overall success of the process. Severe problems may occur, if an activity does not behave in the expected manner. Since workflows may be very complex and activities are in general highly distributed, heterogeneous and autonomous, *advanced recovery concepts* are necessary for adequate failure- and exception handling. WFMSs are expected to support the reliable and consistent execution of workflows, but as has been pointed out in [11], up to now, most WFMSs lack such a functionality. There are only a few notable exceptions, such as IBM FlowMark [22, 3] which offers at least some primitive recovery concepts.

Summing up, the main goal of workflow recovery is, to restore - automatically if possible - the most recent consistent process state after a system or semantic failure so that as few as possible work performed over long-durations is lost and process execution can be continued.

4.1. Recovery after System Failures

A system failure causes a non regular, abnormal termination (abort) of one or more *active activities*, more precisely *active tasks*. If such a situation happens, the WFMS cannot proceed with its regular process execution. Instead, the workflow recovery manager (WRM) has to apply *forward recovery* which comprises crash recovery and forward execution. *Crash recovery* means that all inconsistent execution results of the interrupted and probably half-way executed tasks are removed. Therefore, in most cases some kind of rollback is necessary. In general the WRM is not responsible for a task's internal rollback process because tasks are treated as black boxes and hence it is expected that they are responsible themselves for a correct recovery (they should have their own local recovery system). Since we cannot always presume such an ideal behavior (tasks are not necessarily failure atomic) the WRM should be be able to handle such situations. *Forward execution* means that process execution is resumed from the "closest consistent" point where the failure occurred. This implicitly means that forward execution is applied after crash recovery.

Based on the previous concepts we now investigate specific recovery techniques within workflow execution. This investigation distinguishes between different task types:

- *Automatic tasks:* Forward recovery concerning aborted automatic tasks comprises:
  - Restart of the same task. This is possible if the interrupted task is failure atomic (e.g. a DB-transaction) which means that the task is rolled back automatically or if such a rollback is not necessary because the task is idempotent (e.g. a task which formally checks a loan request form). An idempotent tasks can be executed one or more times without changing the result which is a very comfortable feature within workflow execution.
  - Start of an alternative task. If the same task cannot be restarted after a system failure it may be necessary to start another task instead which removes inconsistent side effects of the interrupted task and which tries to perform the original goal in an alternative way.
  - Manual intervention: If a task is not failure atomic and there exists no alternative task then a manual intervention is necessary. This means for example, that the WRM informs a human agent (for example the process owner) who repairs the failure.
- *Manual tasks:* Forward recovery for manual tasks primarily must be performed by the workflow participant who is responsible for the task (e.g. if a phone call is interrupted then it is the agent who is responsible to take appropriate recovery measures (e.g. to make a new phone call, to restart the word processor and load the file manually)). There are only limited possibilities to automate forward recovery for manual tasks.

To increase the success rate of a correct and automatic task recovery after a system failure the WRM should be able to restart and retry the same (or an alternative) task several times. The number of possible restarts must be defined by the workflow administrator. If a recovery of an interrupted task is not feasible within the specified retry or time limits then the recovery procedure for semantic failures is invoked (i.e. the task fails and *the system failure migrates to a semantic failure*).

4.2. Recovery after Semantic Failures

A semantic or logical failure occurs, if (a) an *activity fails* (commits unsuccessfully) or (b) an authorized agent wants to *undo* (at least parts of) an active process. For the first case, the WRM has to decide whether an inconsistent state is reached or not. Depending on this decision either a complex recovery procedure has to be started or process execution can be continued. The decision whether the failed activity has produced an inconsistent state for the overall process or not depends on the "importance" of the activity in the process. For this purpose, the workflow designer has to define during process definition whether an activity is vital or not. If a vital activity fails then an inconsistent state is reached. This fact, as well as the agent triggered undo, initiates a complex recovery procedure. The main challenge thereby is, to semantically rollback already completed activities - normally in inverse order - until a consistent process state is reached from where the workflow can be continued (by executing an alternative path) or terminated.

4.3. Recovery Concepts for Document-oriented Workflows

For a document-oriented workflow, the reason for recovery is not only a failure within an activity but also a changed situation in the world outside the system (an unexpected exception). Such an exception may for example cause an already approved business-trip to be cancelled later on. The following transaction based exception and failure mechanisms should be available within a document-oriented WFMS (see also [25]):

- Undo: The undo operation allows the agent to undo all work which has been done so far within the (still active) task. There are two problems: first, it is not always possible to undo a task and second, there are only primitive recovery concepts available if the tasks are of manual type (e.g. to restore an old document version).
- Back: The back operation first executes an undo operation if the task is active and then the process is transferred back into the preceding process step (e.g., a document is returned to the agent in charge of the preceding task) with some additional information explaining the reason for the back operation. Now, the new "old" agent has the possibility to update some parts within the task in order to continue process execution afterwards or to use the back operation again.
- Backward recover: Authorized agents should have the possibility to interrupt an *active* process and to delegate the process back into any previous passed state (which is the same semantics than several single back-operations). The backward process comprises the following steps:
  - Visual display of the backward-recover-mode: When a process enters the backward-recover-mode then this should be made explicit visible for at least all involved agents.

– Backward recovery of the process in inverse order: The process is executed in inverse order by semantically undoing previous executed activities. Activities which do not need to be compensated are skipped.

– Decision of the further route: After all required activities have been compensated the agent who initiated the backward recover mode can reactivate the activity in order to add or change some parts. After all work is completed, the agent has to determine the next step in the process, which may be the execution of an alternative path or again a back operation.

– Forward execution: After a back or a backward recover the agent can decide to continue with process execution in a forward direction (normally on an alternative path).

Within the project *"Workflow Transactions"* we currently discuss the integration of at least some of the above presented concepts into the commercial WFMS CSE/Workflow [2].

4.4. Recovery Concepts for Process-oriented Workflows

In general, workflows are structured hierarchically in order to facilitate workflow modeling and to support reuse. Workflows, or in our terminology complex activities, may consist of several logical dependent sub-activities. This means that a complex activity comprises several sub-activities which together determine the success or non success of its parent-activity. A parent-activity succeeds if all, or at least its *vital* sub-activities succeed.

If during process execution an activity fails (commits unsuccessfully) then the WRM has to decide whether the process has reached an inconsistent state or not. This decision is very easy since an inconsistent state can only be reached if an activity fails whose relationship to its parent is vital (in other words: if a vital activity fails). Summing up, after a semantic failure, the WRM has the following possibilities to support a failure tolerant process execution:

- *Forward execution:* If the failed activity has no vital relationship then a positive and consistent termination of the corresponding parent activity can be achieved very easily by making forward progress (ignoring the failed activity) and executing the remaining child activities. The fail of the non vital activity can be tolerated without further consistency preserving measures.

[2]Workflow is a trademark of CSE Systems, Computer & Software Engineering GmbH

- *Backward recovery:* If the failed activity has a vital relationship then a complex recovery procedure is necessary in order to reach a consistent state again. A fail of a vital sub-activity makes a further execution of the remaining child activities obsolete. Instead, a controlled rollback has to be initiated. This means that all previous child-activities which have terminated successfully have do be undone in a correct way. Of course, this is not always possible and sometimes this will require at least human intervention. In [8] the problem of irreversible side effects has been discussed and some solutions for *safe process schemas* have been introduced. After the recovery process at the current level has terminated (and hence all successful committed brother activities of the failed activity have been rolled back semantically), the parent activity will itself terminate unsuccessfully. This, of course, may initiate another backward recovery process at the next higher level.

  Similar concepts concerning backward recovery are discussed in [19]. Within this approach any collection of activities can be defined as a *sphere of joint compensations*, which means that all activities must either run syntactically successful or all activities must be compensated. Since the activities which belong to one sphere may be spread over the whole process and, additionally, spheres may overlap, the compensation process on the one hand seems to be more flexible but on the other hand it seems to be much more complex and hence more difficult to use.

- *Forward recovery* In most cases a combination of backward recovery and forward execution will be applied. Since this steps are very similar to the forward recovery concept for system failures it seems to be reasonable to use the same terminology - forward recovery - within the current context. As soon as a consistent state is reached - a non vital parent activity fails - the WRM will enforce regular process execution, probably along another execution path, in order to make forward progress. WAMO offers choice activities which are an ideal point to change from backward to forward execution. The possibility to undo only *parts* of a workflow is in contrast to many advanced transaction models which always undo complete workflows.

5. Realization Requirements

An integration of transaction-specific features into a WFMS requires a certain functionality from the WFMS. Within this section, we present an overview of the most relevant WFMS requirements which are necessary for the integration of workflow transactions. Of course, not all concepts are required if only a partial integration of workflow transactions is intended.

- *Nested workflow:* For modeling reasons and reusability aspects a hierarchical representation of complex workflows is an important feature. Thereby, a complex workflow is decomposed into smaller sub-workflows (activities) until elementary workflows (called tasks in WAMO) are remaining. Workflows containing other workflows are often called composite workflows. In WAMO such composite workflows are called complex activities. The execution of a hierarchical structured workflow starts at the top-level (most abstract) workflow in the hierarchy by executing the first underlying layer of sub-workflows. Complex activities are logical units of work that determine the co-ordination and data flow requirements between sub-activities. For the realization of transactional concepts, WAMO explicitly demands nested workflows.

- *State of activities:* As mentioned before, within nested workflows it must be distinguished between inner nodes (activities) and leaf nodes (tasks). The main difference between these types is that activities are *fully controllable* by the WFMS (by the transaction manager and/or scheduler) whereas tasks are only *partially controllable* (similar ideas are discussed in [18]). This means, that for example the state of an activity can be set to commit successfully by the workflow transaction manager as soon as all sub-activities have terminated regularly, whereas the state of an already started task can only be determined by the corresponding processing entity. In order to enable transaction based workflow execution, the existence of different states of activities and tasks is necessary. The event state diagram of activities and tasks in WAMO are illustrated in figure 3 and 4.

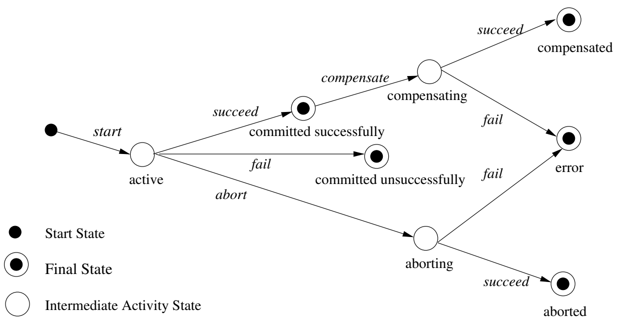

Figure 3. Event-state diagram for activities

An activity can be started if it is in the initial state startable. The start event changes the state of the activity to active. Now the corresponding child-activities are executed. After the child-activities have finished the activity terminates. In the regular case the activity will either commit successfully (succeed) or commit unsuccessfully (fail). These termination states are fully controlled by the workflow controller. The result, of course, depend on the termination states of the child-activities. Besides the regular case it is also possible that an activity is aborted by an external event (e.g., by the user) or by the system. Then the state of the activity is changed to aborting which means that all active child-activities are aborted and all successfully committed child-activities are semantically rolled back (compensated). Since a semantic roll back must be supported within transaction-based workflows, activities are equipped with the corresponding compensation concepts (for further details see [8]).

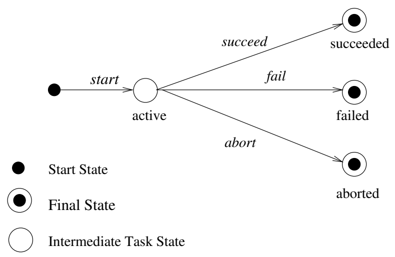

Figure 4. Event-state diagram for tasks

Tasks are elementary activities which are directly executed by an processing entity. Hence, the workflow controller can only start a task but not determine the execution result of the task. In general, a task will either succeed in the sense of *commit successfully* or fail in the sense of *commit unsuccessfully*. Additionally, a task can terminate abnormally (abort). Since the WRM has to react according to these different termination states a corresponding distinction is necessary.

- *Restart information for tasks:* After a system failure, a task may be in an inconsistent or undefined state, which means that uncommitted side effects could exist somewhere in the system. The WRM should be able to remove all inconsistencies and to resume process execution from the nearest consistent point where the failure occurred. Therefore, the WRM needs the following information which has to be specified by the workflow designer during process modeling time:

  – Which task has to be started after an abnormal task termination (e.g., the same, an alternative one or the next in the sequence)?

  - How often should a task be restarted after a system failure?
  - Is a manual intervention necessary?

- *Compensation tasks:* In case of backward recovery it may be necessary to compensate (semantically undo) already committed tasks. For that reason, WAMO introduces the task specific *storno-type parameter*. The idea is, to associate tasks with corresponding *"compensation (inverse) tasks"* [17] which are executed in case the original task has to be undone semantically. But there are also other kinds of compensation which have been explained in subsection 2.1. As with the original tasks, also compensation tasks should be written in such a way, that they can easily be reused in other workflows. Additionally, it must be emphasized, that it is much easier to define compensation tasks (respectively to generate them automatically) if structured data (see subsection 3.1 is manipulated within the workflow.

  Within WAMO, up to now only tasks are associated with the corresponding compensation tasks. Complex activities are only responsible for a correct compensation of their child activities but the model can be extended easily in that way, that also complex activities have their own (complex) compensation activities. This concepts are similar to the idea of discrete and integral compensation in [20].

  In order to guarantee a consistent recovery process, it is necessary that compensation tasks terminate successfully otherwise manual intervention is required.

- *Initiation of the recovery process:* A recovery process is either initiated manually by a human agent or automatically because of a system or logical failure. A *manual initiation* is in general triggered by activating the back or backward recover function. Since these functions have a major impact on the future workflow execution, appropriate authorization rules are necessary. This means, for example, that only specific agents are authorized to use the backward recover function and / or that the authorization profile changes dynamically during process execution. An *automatic initiation* of the recovery process is triggered after a system failure, a logical failure or a user cancel command.

- *Controlling of the recovery process:* The controlling of the recovery process is a central topic within a transaction-based workflow execution. The inverse execution of a process during backward recovery is as important as the forward execution of a business process. A backward recovery process comprises the following features:

  - Initiation of the backward recovery process as explained before.
  - The backward execution path is based on the already executed path. In general, the backward path is the inverse of the forward path. Therefore a process history is absolutely necessary. This means, for example, that the execution states of all activities in the workflow have to be stored persistently. Tasks with the storno type "none" can be skipped during backward execution.
  - The compensation tasks have to be provided with the proper application and process data.
  - The backward process terminates as soon as the closest consistent point is reached.

- *Logging:* As already mentioned before, a transaction-based workflow execution demands extensive logging activities. Of course, every WFMS does some logging and some of this information can be reused within a transaction-based execution. The data which is logged should be kept in the workflow database. Logging comprises two main areas:

  - Logging of process data: For the complex recovery procedure the WRM needs the information of the state of all activities (and tasks), the execution history and the agents who have performed the various activities. Normally, most of this information is already gathered by the system for process monitoring and tracking.
  - Logging of application data: Besides the control flow aspects also the data flow within a workflow is of major interest for the recovery process. Compensation activities have to be provided with the proper data in case of an activation. We can distinguish between the following data extensions which are necessary for a compensation activity: (1) data, which is explicitly added by the user for the purpose of the compensation process (especially within document-oriented workflows), (2) input-data of the original activity and/or of several other activities, and (3) output-data of the original activity and/or other activities.

6. Conclusions

Workflow management systems more and more become the basic technology for organizations to perform their daily business processes (workflows). These processes tend to be of long duration, involve many users and tools over heterogeneous and distributed environments. We claim that a consistent and reliable execution of such workflows can only be achieved by integrating transactions - *workflow transactions* - into WFMSs. The main difference between traditional database transactions and workflow transactions is the fact, that the goal of database transactions is to transform *data* from one consistent state into another consistent state in the presence of failures and concurrent access, while workflow transactions aim to transform *business processes* from one consistent state into another consistent state.

Based on the idea of workflow transactions, we have discussed in detail advanced workflow recovery concepts which are necessary for a reliable and consistent execution of workflows in the presence of failures and exceptions. Therefore, we have analyzed different failure sources and failure classes which influence and determine the recovery concepts. Additionally, we distinguish between two main classes of workflows, document-oriented and process-oriented, because they have different requirements for workflow recovery.

References

[1] G. Alonso, R. Günthör, D. Agrawal, A. E. Abbadi, and C. Mohan. Exotica/fmdc: Handling disconnected clients in a workflow management system. In *Proc. of the 3rd Int. Conference on Cooperative Information Systems*, Vienna, Austria, May 1995.

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

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

[4] 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.

[5] A. Buchmann, T. Oezsu, M. Hornick, D. Georgakopoulos, and F. Manola. A transaction model for active distributed object systems. In A. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, 1992.

[6] C. Bussler. Policy resolution in workflow management systems. *Digital Technical Journal*, 6(4):26–49, 1994.

[7] 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.

[8] 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.

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

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

[11] 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.

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

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

[14] S. Jablonski. Functional and behavioral aspects of process modelling in workflow systems. In G. Chroust and A. Benczur, editors, *CON 94 Workflow Management: Challenges, Paradigms and Products*. R. Oldenburg, 1994.

[15] W. Jin, M. Rusinkiewcz, L. Ness, and A. Sheth. Concurrency control and recovery of multidatabase work flows in telecommunication applications. In *SIGMOD*, May 1993.

[16] M. Kamath and K. Ramamritham. Modeling, correctness & system issues in supporting advanced database applications using workflow management systems. Technical report, University of Massachusetts, 1995.

[17] H. Korth, E. Levy, and A. Silberschatz. A formal approach to recovery by compensating transations. In D. M. et al., editor, *Proc. of the 16th Int. Conference on Very Large Data Bases*, Brisbane, Australia, 1990.

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

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

[20] F. Leymann. Transaktionskonzepte für workflow managemen systeme. In G. V. J. Becker, editor, *Geschaeftsprozessmodellierung und Workflow-Management*. Thomson, Germany, 1995.

[21] S. McCready. There is more than one kind of workflow software. *Computerworld*, 2, Nov. 1992.

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

[23] M. Rusinkiewicz, A. Elmagarmid, Y. Leu, and W. Litin. Extending the transaction model to capture more meaning. *ACM SIGMOD Record*, 19(1), Mar. 1990.

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

[25] P. Vogel and R. Erfle. Backtracking office procedures. In A. M. Tjoa and I. Ramos, editors, *Proc. of the Int. Conference on Database and Expert System Applications (DEXA 92)*. Springer-Verlag, 1992.

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