Contributions to Exception Handling in Workflow Management

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

Abstract

We propose a concept for exception handling in workflow management systems. An exception arises if an activity of the workflow does not deliver the desired results due to continuing system failures, environmental circumstances or inadequacy of a business process in some situation. The concept we propose consists mainly of automatic compensation and pursuing alternatives and integration of semi-automatic at hoc intervention. The goal was to ensure consistent execution of business processes with little extra effort at build time as well as at runtime.

1 Introduction

Workflow management systems [GHS95, Law97, WfM96] support the execution of business processes. Workflows are technical representations of business processes which are enacted, controlled and documented by a workflow management system. A workflow typically consists of steps called activities or tasks and the dependencies between these steps. An activity represents the performance of some piece of work by an user and/or an IT-system.

As all IT-systems, workflow management systems have to care for failures and exceptions. By *failures* we mean that an IT-system does not work as expected (e.g. program error, device failure, communication failure, etc.). In contrast, *exceptions* (sometimes also called *semantic failures*) arise when activities cannot be executed as planned or do not deliver the desired results. Exceptions may be caused by unrecoverable system failures or by some situation in the environment. Examples of exceptions are:

- A customer has to be called, but he does not answer the phone.
- A flight should be reserved, but the plane is fully booked.
- A program cannot be executed because it has errors.
- Money should be withdrawn from an account but the balance is negative.
- Money should be transfered to an account but the bank got bankrupt.
- The business process demands that billing is made before shipping but an important customer with high reputation wants it the other way round.

From these examples you see that some exceptions may occur very frequently and are usually treated explicitly in the business process or workflow model. Other exceptions might occur very rarely and may be not thought of at design time, and then there are exceptions which cannot be dealt with at design time because they arise as consequences of changes in the environment (like new laws, regulations, policy decisions from the management).

Reactions to exceptions can be:

- *retry:* Repeating the failed activity might help in some cases, but obviously this can only be applied a restricted number of times.
- *ignore:* A solution, if the activity is not necessary for the overall success of a business process
- *rollback the business process:* If a business process cannot be executed to a successful end, it has to brought to a consistent end state. This may mean to free resources (e.g. budgets, personal) assigned to the process, cancel reservations made, inform people, and so on.
- *partial rollback and forward execution:* We propose mainly this reaction. It means to roll back the process to some point where a decision of the continuation was made and proceed along an alternative path.

- *process the business case outside of the workflow system:* This reaction is always possible but should be avoided for obvious reasons.
- *find new way to overcome the problem:* This means to change the business process and accordingly the workflow dynamically, either as individual workflow for a certain (single) business case or as a new version of the business process (resp. workflow).

Common approaches to deal with exceptions in workflow management systems are explicit treatment of exceptions in the definition of workflows or ad hoc treatment at run-time within or outside of the workflow management system. In the first approach, the designer of a workflow takes into account that any activity could fail and takes measures to react to such a failure after each activity. This resembles the task of a programmer of transactional software who also has to check after each database access whether there was no failure, rollback, etc. The problem with this strategy is that the number of paths in a workflow will explode making the workflow hard to understand and maintain. Moreover, this strategy does not help in situations where the business process itself has to be changed. On the other extreme, failures are not considered at build time and exceptions are treated in a pure ad hoc manner or even outside of the workflow management system. Obviously, these strategies are more complex at run time and even may corrupt the workflow system as it no longer has information on all instances of a certain business process and does not represent the actual state of business processes.

In [EL94, EL95, EL96] we introduced a workflow model which assists the designer with an automatic failure handling feature which is mainly based on partial rollbacks to explicit decision points (choices) in the workflow. From these points forward execution of further paths is triggered. Based on the experience that decisions are not only made at in explicit choices but within activities we extend the model such that in principle any activity could be redone and be the turning point from compensation to forward execution. Additionally we provide different levels of workflow recovery which gives the designer the option to change the general recovery process where it is not appropriate. Furthermore we want to make the exception handling mechanism so flexible that any exception can be dealt with within the system. So we integrate an ad hoc mechanism which can be invoked at any state of the workflow and enables process managers to handle the exceptional business case with some extensions or alterations of the workflow up to a dynamic change to an individual workflow.

Up to now, failure and exception handling in workflow management has not been investigated deeply [GHS95, KR95, AAAM96, WS97]. System failures have been intensively studied within the classical transaction model [GR93] and to some extend within workflow management. Especially within the Exotica project [MAGK95] different types of system failures within a distributed workflow environment have been discussed. Also extended transaction models [Elm92] and transactional workflows [SR93] primarily focus on handling system failures. Some of them also support the explicit modeling of failures and exceptions (e.g. [WR92]).

Semantic failures (exceptions) are investigated partially within the Excotica and the METEOR$_2$ [WS97] project. However, they either suggest the explicit modeling of exceptional situations or they propose human intervention in case of exceptional situations.

2 Basic Concepts

To support exception handling within workflow management some basic pre-conditions are necessary. First, the workflow metamodel must be extended in order to incorporate exception-specific information. Second, the workflow enactment service (workflow engine) must be able to execute workflow instances based on such an extended workflow schema. We will discuss these basic concepts by introducing the *Workflow Activity Model WAMO* [EL94, EL95, EL96] which was developed in order to support not only the handling of system failures but especially of exceptions within workflow management. In *WAMO* a workflow has a hierarchical structure, consisting of complex activities, elementary activities and tasks.

2.1 Task

Tasks are the atomic work items which are executed by the workflow participants. We distinguish between automatic and manual tasks. Automatic tasks are either batch programs which are executed without any user interaction or interactive workflow applications. Manual tasks are work items which are not executed with a specific workflow application (e.g. making a phone call, writing a letter). For the handling of failures and exceptions two aspects are important: *Task state information* and *task recovery information.*

Tasks have a complex internal structure, i.e. different states which are reached by special events. In figure 1 the state diagram of a task in *WAMO* is presented. Besides the usual states `ready`, `taken` and `active` the termination states of tasks are more

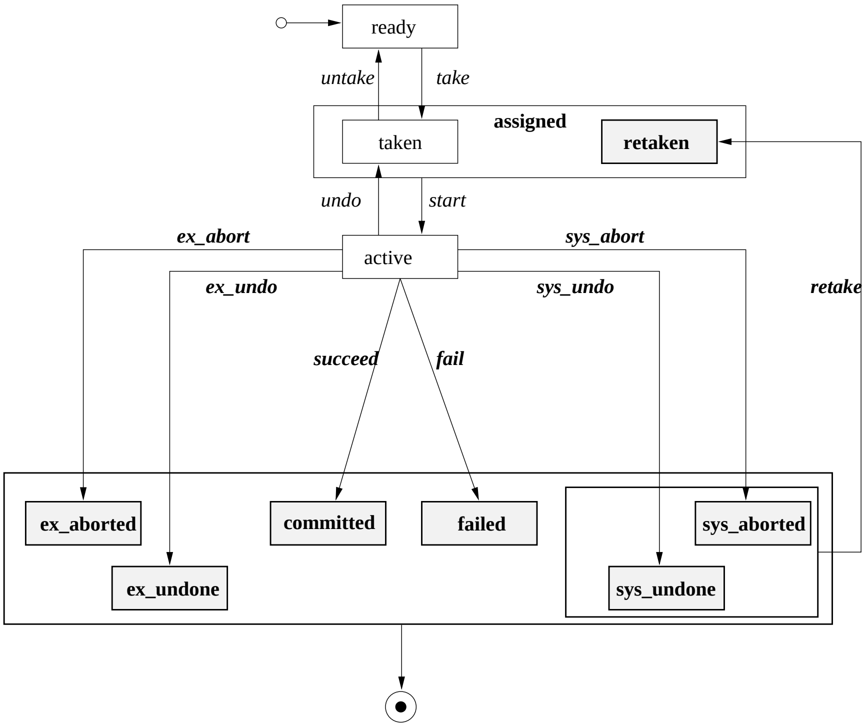

Figure 1: Extended state diagram of a task

extensive in order to detect whether a failure has occurred. We distinguish between three different classes of termination:

1. *regular termination:* This class comprises the general termination states of a task, namely `committed` (terminated with a positive result) and `failed` (terminated with a negative result).

2. *controlled abort:* The second class consists of states which are reached after a controlled cancelation of a task by the workflow user or the workflow system (e.g. task "`car reservations`" is aborted because the customer does not need a car any longer). Depending whether the task is failure-atomar or not the termination states after an abort are `ex_undone` or `ex_aborted`. The event `ex_undo` causes an internal rollback of the task, the event `ex_abort` terminates a task without cleaning up any intermediate results. Additionally, it must be emphasized that not all tasks are abort-able in a controlled way. Therefore, a task within $WAMOs$ metamodel contains the attribute "`cancelType`" which may have the values `ex_abortable, ex_undoable` or `no-cancel` and which must be set by the task programmer.

3. *uncontrolled abort:* Complementary to the previous cases, tasks may be terminated uncontrolled, because of system failures. Again, we have to distinguish between the states `sys_undone` (for failure atomar tasks) and `sys_aborted` (we also refer to these failure states as *sys_terminated*). Remember that the attribute `cancelType` contains the necessary failure atomicity information of a task which will be necessary for further failure handling.

As an example for these different task termination states we analyze an automatic teller machine and the task ATM-transfer:

task *ATM-transfer*
    `cancelType[ex_undone, sys_undone]`

To withdraw money there must be enough money on the account and the PIN-code must be correct. In this case the task *ATM-transfer* terminates in the state `committed`. If there is not enough money on the account or the PIN is wrong then the task fails. Additionally, during money transfer the customer may cancel the operation which leads to the state `ex_undone`, i.e. the operation is rolled back. If during task execution a system failure occurs then again an automatic rollback must be guaranteed such that neither the bank nor the customer is aggrieved. In that case the task terminates in the state `sys_undone`.

2.2 Elementary Activity

The concept of *compensation* is essential within *WAMO*. The first version of *WAMO* [EL94, EL95, EL96] enabled the workflow designer to associate tasks with compensation tasks. Now this concept has been refined and enhanced by more sophisticated compensation techniques (see also [Ley95a, AAA+96]). First, not only tasks but also complex activities may be associated with compensation activities (i.e. tasks or again complex compensation activities). Second, a new building block - *elementary activities* - are introduced (see figure 2).

An elementary activity contains a task and the corresponding compensation task, if available and necessary. As already mentioned in previous work, *WAMO* does not presume that every task has an associated compensation task (as this is for example the case in the Saga-Model [GMS87]). Sometimes a compensation is not possible (e.g. faxing a confidential information to the wrong person) and sometimes a compensation is not necessary (e.g. an automatic task which checks the credit-worthiness of a customer). Elementary activities have all the features of tasks but also further information concerning the compensation of a task. In particular, they offer two types of compensation:

1. *strong_compensation*
2. *weak_compensation*

If an elementary activity receives the event `strong_compensate` then the task within the activity is compensated according to *WAMOs* compensation technique. On the other hand, if a `weak_compensate` event is received then within the elementary activity the decision is made, whether the task is compensated or the compensation is finished (inverted) by executing the original task again. This concept opens two important advantages: first, it simplifies the specification of workflows since elementary activities represent black boxes for the workflow designer and second it allows a much more flexible handling of the compensation process since depending on the compensate event, a compensation process can be finished precociously.

2.3 Complex 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. 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*, *and-parallel*, *nesting*, *loop*, *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).

As already mentioned, complex activities may be associated with compensation activities (i.e. tasks or again complex activities). However, this opens two possibilities for the compensation of a complex activity: shallow and deep compensation (a similar approach is introduced in [Ley95b]). Within *shallow compensation* the associated compensation activity is executed whereas *deep compensation* propagates the compensation of a complex activity to its child activities. The attribute `compType` of a complex activity must be set by the workflow designer in order to define which compensation type should be taken at runtime within a given workflow type. The state diagram of a complex activity is presented in figure 3.

In contrast to tasks, complex activities *cannot* be aborted uncontrolled. The state of an activity is computed by the states of its child activities. As soon as a child terminates the event `Cterm(termination_state)` is sent to the parent activity (see figure 3). Hence, we basically distinguish between two classes of termination states:

1. *regular termination* comprises the states `committed` (terminated with a positive result) and `failed` (terminated with a negative result).
2. *controlled termination* either belongs to an external abort (state `ex_aborted`) or to a compensation (state `compensated`). An external abort may occur when the whole workflow is aborted by the user. A compensation (weak or strong) if the activity is involved in a compensation process.

2.4 Vitality

Another important basic feature is the concept of *vital* and *non vital* activities in order to specify the importance of (complex) activities within a workflow. During workflow modeling the workflow de-

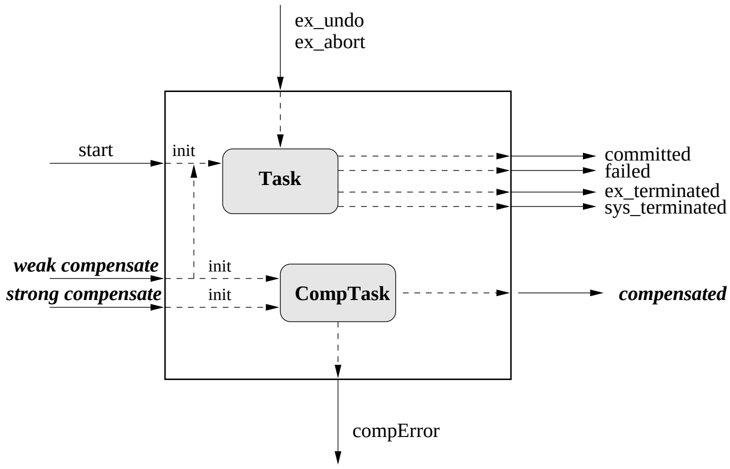

Figure 2: Internal structure of an elementary activity

signer specifies which child activities are vital or non vital for the corresponding parent activity. Therefore, $WAMOs$ metamodel offers the relation attribute `vital` which can be set for any activity.

If a non vital activity fails, workflow execution can continue and make forward progress without any compensation actions. Per default, all activities within a workflow are 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 3).

3 Exception Handling

3.1 Automatic Exception Handling

At workflow execution time a task (and hence an elementary activity) may fail either because of a controlled abort (cancel by the user), an uncontrolled abort or a regular fail. In [EL96] several techniques concerning the handling of system failures (uncontrolled abort) of tasks have been presented: especially the *retry* of tasks or the execution of alternative tasks. Additionally, an escalation concept is discussed, which supports the *migration* of system failures to semantic failures, i.e. if the system failure cannot be handled at task level then the failure escalates to an exception. As already mentioned, exceptions may also arise if a task does not deliver a desired result or cannot be executed because of unpredicted changes in the business process.

In such a case, further execution depends on the vitality of the failed (elementary) activity. A complex activity terminates in the state `committed`, if all activated vital subactivities terminate successfully. Otherwise the complex activity runs into a compensation procedure. This leads to the following execution decisions:

1. *Forward execution:* If the failed activity A_i is *non vital* then the workflow continues.
2. *Backward recovery:* If the failed activity A_i is *vital* then the compensation process is activated in order to re-establish a consistent workflow state. The execution sequence within a compensation depends on the current control structure A_i is participating:
   (a) *sequence:* The state of the parent PA_i of activity A_i is changed to `weak_compensating` (see event `Cterm[vitfail]` in figure 3). The direct predecessor[1] A_{i-1} of A_i within PA_i is searched.
      i. If there is no direct predecessor under the parent PA_i then the parent terminates in the state `compensated`, which either finishes the compensation process if the parent is a non vital activity or terminates the whole workflow in the state `compensated` if there is no parent or continues with the compensation of the predecessor of PA_i.
      ii. If there is a direct predecessor then the last computed elementary activity of this direct predecessor A_{i-1} is

[1]The determination of the direct predecessor can be fairly complex since several traversals within the activity hierarchy are necessary. A prototype implementation of this mechanism is realized in the prototype WFM-System *Panta Rhei*[EGL97] and described in [Lie98].

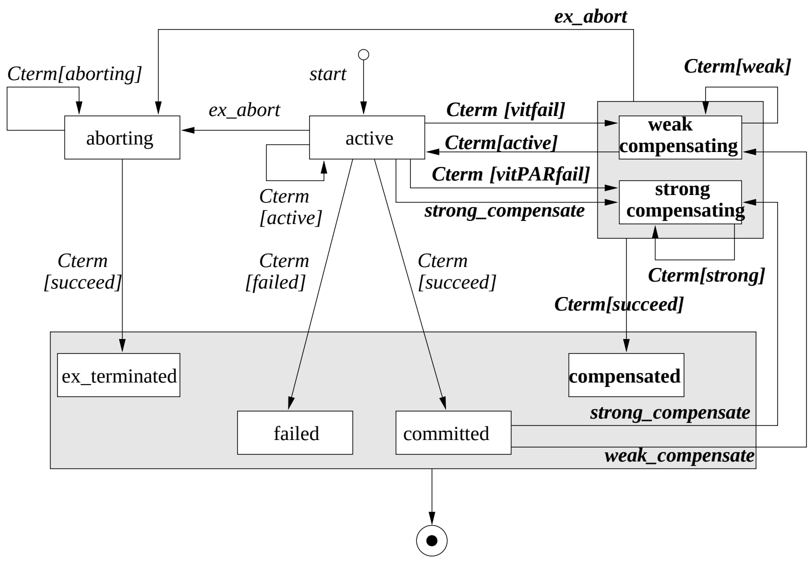

Figure 3: State diagram of a complex activity with compensation

searched. However, this again may require several traversals within the activity tree if A_{i-1} is a complex activity.

iii. If A_{i-1} is an elementary activity then it is compensated through the method `weak_compensate`. This means that the task within the elementary activity is compensated and the activity terminates in the state `compensated`. Or compensation is finished precociously, i.e. the elementary activity terminates successfully. In this case, the state of the parent activity changes from `weak_compensating` to `active` (event `Cterm[active]`).

(b) *parallel:* If a vital parallel activity A_i fails then already executed siblings of the activity are *strong* compensated. The state of PA_i is changed to `strong_compensating` (see event `Cterm[vitPARfail]` in figure 3). As soon as all siblings are compensated, PA_i changes its state to `compensated` which either finishes the compensation process if PA_i is a non vital activity or terminates the whole workflow in the state `compensated` if there is no parent of PA_i or continues with the compensation of the predecessor of PA_i according to the previous discussed algorithm.

(c) *choice:* The activities within a choice are all non vital. If an activity within the choice fails then the next alternative within the choice is activated. The choice activity itself A_i fails if all activities within the choice structure fail. This may trigger a compensation process if the failed node is vital. The compensation depends on the control type of the parent of A_i.

If during the compensation process a complex activity of type choice receives a (weak or strong) compensate event, then this event is applied on the successfully terminated activity within the choice.

(d) *loop:* The compensation of a loop depends on the attribute `compType` which must be set by the workflow designer during workflow build time. If the attribute value is of `full compensate` than each successful executed activity within the loop is compensated (regardless how often the loop was executed). If the value is of `partial compensate` then only the last iteration is compensated.

3. *Forward recovery:* Having re-established a consistent state by doing backward recovery, workflow execution either finishes because all committed activities have been compensated or continues if:

   (a) there is an alternative path (e.g. an other activity in the choice),

   (b) one of the parents of the compensated activity is non vital,

(c) the compensation was inverted within an elementary activity.

3.2 Ad hoc Exception Handling

Although the concept of weak compensation offers a flexible mechanism to handling the compensation process within a workflow we still do need further possibilities to support workflow users in handling exceptional situations. During any stage of workflow execution workflow users should have the possibility to leave the predefined workflow path, execute some additional activities, jump back into the predefined schema and continue with workflow execution. Therefore, two types of ad hoc executions are introduced within $WAMO$:

1. *ad hoc extension:* By applying this function during execution of a task, the workflow user can jump into the ad hoc mode if the current state of the task is `taken` or, if the current state is `active`, as soon as the task terminates. Jumping into the ad hoc mode comprises the following steps:

   (a) The workflow user must select the new activity and the agent of this new activity. Additionally, he must be able to supply the new activity with necessary input data.

   (b) The workflow engine creates a new version of the schema if the ad hoc mode is activated the first time within the current instance. Otherwise the already extended version is used.

   (c) After execution of the included activity, workflow execution automatically continues with the next activity in the schema or with another new activity if he user again activates the ad hoc mode.

2. *ad hoc refinement:* In contrast to the previous ad hoc function the idea of ad hoc refinement is to interrupt current task execution in order to execute in the meantime one or more new activities and then to continue with the interrupted task. However, this function can only be applied on tasks which are suspendable. This feature is set by the workflow designer (attribute `suspendable` within a task specification). The actions within an ad hoc refinement are similar to these of an ad hoc extension.

4 Conclusions

We proposed a concept of failure handling in workflow management systems which offers several desirable properties. First it treats business processes as business transactions and enforces that workflows terminate in a correct state. The backward recovery feature automatically drives the compensation of activities. This failure handling model eases the task of designers as well as of workflow agents and process managers. Designers may use its compensation/forward execution model to automatically generate a series of consistent alternative execution plans for a model and need not treat any exception explicitly. Compensation and forward execution are enacted automatically so the need for intervention and decision making in the cause of exceptions at run time is reduced. Through the integration of ad hoc semiautomatic exception handling, the system is very flexible and gives process managers a whole range of functionalities to treat any exception within the workflow management system. The automatic exception handling functionality comes with little extra effort at design time where information about activities have to be given to the system. We described the proposed exception handling concept in terms of our *Workflow Activity Model* $WAMO$, however the concept is more general and can easily be integrated in other workflow specification models.

References

[AAA+96] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Günthör, and C. Mohan. Advanced Transaction Models in Workflow Contexts. In *Proc. of 12th IEEE Intl. Conference on Data Engineering*, New Orleans, LA, 1996.

[AAAM96] G. Alonso, D. Agrawal, A.El Abbadi, and C. Mohan. Functionality and Limitations of Current Workflow Management Systems. *IEEE Expert Journal*, 1996.

[EGL97] J. Eder, H. Groiss, and W. Liebhart. The Workflow Management System Panta Rhei. In A. Dogac et al., editor, *Advances in Workflow Management Systems and Interoperability*. Springer, Istanbul, Turkey, August 1997.

[EL94] J. Eder and W. Liebhart. A Transaction-Oriented Workflow Activity Model. In S. Kuru et al., editor, *Proc. of the 9th*

Int. Symposium on Computer and Information Sciences, pages 9–16, Antalya, Turkey, 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.

[EL96] J. Eder and W. Liebhart. Workflow Recovery. In *1st IFCIS Int. Conference on Cooperative Information Systems*, Brussels, Belgium, June 1996. IEEE Computer Society Press.

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

[GHS95] D. Georgakopoulos, M. Hornick, and A. Sheth. 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.

[GMS87] H. Garcia-Molina and K. Salem. Sagas. *ACM SIGMOD Conf.*, 1987.

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

[KR95] M. Kamath and K. Ramamritham. Modeling, Correctness & System Issues in Supporting Advanced Database Applications Using Workflow Management Systems. Technical report, University of Massachusetts, 1995.

[Law97] P. Lawrence, editor. *Workflow Handbook 1997.* John Wiley, 1997.

[Ley95a] 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, March 1995. Springer Verlag.

[Ley95b] F. Leymann. Transaktionskonzepte für Workflow Management Systeme. In G. Vossen J. Becker, editor, *Geschaeftsprozessmodellierung und Workflow-Management.* Thomson, Germany, 1995.

[Lie98] W. Liebhart. *Fehler- und Ausnahmebehandlung im Workflow Management.* PhD thesis, Department of Informatic-Systems, University Klagenfurt, January 1998.

[MAGK95] C. Mohan, G. Alonso, R. Günthör, and M. Kamath. Exotica: A Research Perspective on Workflow Management Systems. *IEEE Data Engineering Bulletin*, March 1995.

[SR93] A. Sheth and M. Rusinkiewicz. On Transactional Workflows. *Bulletin of the Technical Committee on Data Engineering*, 16(2), 1993.

[WfM96] Workflow Management Coalition, Technical Report, http://www.aiai.ed.ac.uk/WfMC/overview.html. *Coalition Overview*, 1996.

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

[WS97] D. Worah and A. Sheth. Transactions in Transactional Workflows. In S. Jajodia and L. Kerschberg, editors, *Advanced Transaction Models and Architectures.* Kluwer Academic Publishers, 1997.