Philadelphia, Pennsylvania, USA, May 31 - June 2, 1999.

Concurrency Control and Recovery in Transactional Process Management*

Heiko Schuldt **Gustavo Alonso** **Hans-Jörg Schek**

Institute of Information Systems
Swiss Federal Institute of Technology (ETH)
ETH–Zentrum
CH–8092 Zürich, Switzerland
{schuldt,alonso,schek}@inf.ethz.ch

Abstract

The unified theory of concurrency control and recovery integrates atomicity and isolation within a common framework, thereby avoiding many of the shortcomings resulting from treating them as orthogonal problems. This theory can be applied to the traditional read/write model as well as to semantically rich operations. In this paper, we extend the unified theory by applying it to generalized process structures, i.e., arbitrary partially ordered sequences of transaction invocations. Using the extended unified theory, our goal is to provide a more flexible handling of concurrent processes while allowing as much parallelism as possible. Unlike in the original unified theory, we take into account that not all activities of a process might be compensatable and the fact that these process structures require transactional properties more general than in traditional ACID transactions. We provide a correctness criterion for transactional processes and identify the key points in which the more flexible structure of transactional processes implies differences from traditional transactions.

1 Introduction

In conventional databases, concurrency control and recovery are well understood problems. Unfortunately, this is not the case when transactions are grouped into entities with higher level semantics, such as *transactional processes* [Alo97]. Some initial work has been done in this direction: studying atomicity (spheres of joint compensation [Ley95], or flexible transactions [ELLR90, ZNBB94]) in a single process and analyzing concurrency control without considering recovery [AAHD97]. Practical experience, however, shows that concurrency control and recovery are related problems and they both need to be solved in order to produce complete, feasible solutions.

In this paper, we present a first attempt to develop a theoretical framework in which to reason about concurrency control and recovery in transactional processes. The challenge we face is to design a single correctness criterion accounting for both concurrency control and recovery which, at the same time, copes with the added structure found in processes. In particular, and unlike in traditional transactions, processes introduce flow of control as one of the basic semantic elements. Thus, the correctness criteria must take into consideration that processes already impose ordering constraints among their different operations and among their alternative executions, constraints that will play a significant role in determining how process execution can be interleaved. Similarly, processes integrate invocations to applications with different atomicity properties. Therefore, we cannot impose the strong requirements used in other models (like ConTracts [WR92, RSS97], or CREW [KR98] where the inverses of all process steps have to exist).

The contribution of the paper is threefold. First, it clarifies the problem of concurrency control and recovery in transactional processes without making unreasonable assumptions about their environment. Second, starting with the correctness of a single process based on flexible transactions [ELLR90, ZNBB94] it provides a correctness criterion for concurrent execution of several processes generalizing and adapting the unified unified theory of concurrency control and recovery [SWY93, AVA+94, VHYBS98] to transactional processes thereby extending the applicability of these models. In contrast to other approaches proposing a variety of transaction models (like TSME [GHS95, GHKM94]), this paper provides a single model covering all requirements that arise in the application areas of transactional process management. Third, it discusses several realistic environments where these ideas are being implemented. We believe that transactional processes are becoming more and more important in applications such as, for instance, electronic commerce or virtual enterprises, workflow management systems, process support systems, or specialized coordination tools. Therefore, we expect the results of this paper to be of practical relevance in a variety of applications.

The paper is organized as follows: In section 2, we present a sample application scenario for transactional processes. In section 3, we develop a correctness criterion for transactional processes and discuss its impact on concurrency control and recovery. Section 4 concludes the paper.

2 Motivation

Computer Integrated Manufacturing (CIM) environments are a good example of the use of transactional processes to coordinate different subsystems [NSSW94]. In the exam-

*Part of this work has been funded by the Swiss National Science Foundation under the project WISE (**W**orkflow based Internet **S**ervices) of the Swiss Priority Programme "Information and Communication Systems".

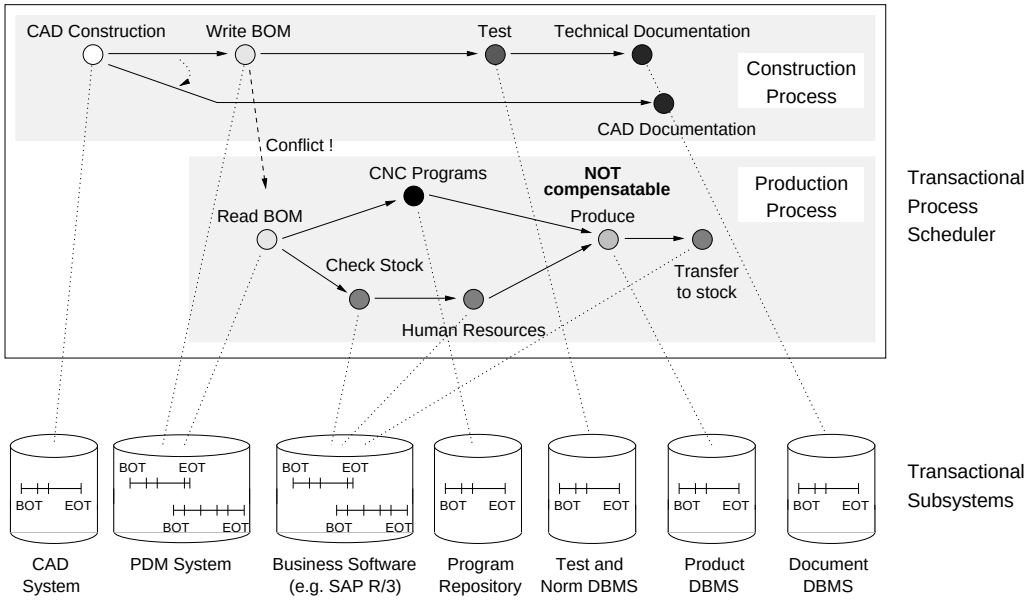

Figure 1: Concurrent execution of a construction process and a production process in the CIM scenario

ple shown in figure 1, two processes are used to control the development and production of new products[1]. In this case, production does not follow mass-production techniques but aims to customize each one of the products to deliver. Thus, the development of the product and its manufacture are strongly tied. The construction process contains all developing steps from the design of a new part to the final test and the subsequent technical documentation. It encompasses a CAD system, a product data management system (PDM), a test database as well as a technical documentation repository. The production process includes all manufacturing steps from the ordering of materials to the production floor including the necessary scheduling. Thus, the production process encompasses the PDM system, a business application, a program repository and a product DBMS. Activities of transactional processes are service invocations in these underlying subsystems. As the bill of materials (BOM) of a new product generated within the construction process provides the necessary input required by the production process, dependencies between both processes exist.

2.1 Extending the Notion of Atomicity

The example above clearly shows why transactional processes must provide a more general notion of atomicity than traditional transactions. Consider the construction process in figure 1. If a failure is detected during the test activity of this process, it is certainly not desirable to undo all previous work including the long running design activity. It is more appropriate to undo only the PDM entry and document the CAD drawing so as to facilitate later reuse. This documentation can be alternatively executed instead of the technical documentation of the whole part which would have been done if the test activity would have succeeded. The possibility of executing alternatives in case of failures therefore generalizes the all-or-nothing semantics of atomicity and leads to a more flexible notion of atomicity used for transactional processes.

[1]This example reflects the practice followed by one of our industrial partners in a recently concluded research project [SST98].

2.2 Concurrency and Interference

An additional prerequisite is to guarantee consistent interaction between processes. Consider a construction process and a production process being executed in parallel as depicted in figure 1. This parallelization is important in practice as it dramatically reduces the time to market of new products. As depicted in figure 1, only the two activities within the PDM system do conflict. For concurrency control purposes, the ordering of these two activities would be sufficient. However, when recovery has to be considered, further dependencies exist. As no inverse for the production activity exists, it must not be executed before the test terminated successfully. If the test fails, the PDM entry is compensated within the construction process and the BOM read by the production process is invalidated. Therefore, all activities of the production process would have to be compensated, too. However, if production of parts is already performed, this would lead to severe inconsistencies as no valid construction and BOM of these parts exists.

2.3 Transactional Subsystems

A transactional process scheduler coordinates transactional processes on top of transactional subsystems and ensures correctness even in case of failures. We assume these subsystems to have functionality such as the atomicity of service invocations, and either the ability to compensate already committed services or to support a two phase commit protocol. When the application does not provide such functionality, it will be provided by wrapping this application system with a transactional coordination agent. In this paper we concentrate on transactional process management on top of such transactional, possibly agent-wrapped subsystems. The problem of wrapping these systems by transactional coordination agents is important but beyond the scope of this paper. Some aspects of this problem are discussed in [NSSW94, SST98].

3 Concurrency Control and Recovery in Transactional Processes

In the following, we will consider transactional processes executed by a transactional process scheduler on top of subsystems supporting the execution of local transactions as shown in figure 1. In this section, we derive a correctness criterion to reason about correct concurrency control and recovery of these transactional processes in a single framework.

3.1 Process Model

Each subsystem provides a limited set of transactional services that can be invoked by processes. Let $\hat{\mathcal{A}}$ be the set of services (*activities*) provided by all subsystems. For each invocation of an activity of $\hat{\mathcal{A}}$, return values are provided. As activities are itself transactions in the underlying subsystems, they are by definition atomic and therefore terminate either committing or aborting. Activities differ in terms of their termination guarantees: they are either *compensatable, retriable,* or *pivot* (as in the flex transaction model [MRSK92, ZNBB94]). In the case of compensatable activities, a compensation service is provided by the underlying subsystem, retriable activities are guaranteed to successfully terminate after a finite number of invocations, and pivot activities are those which are neither compensatable nor retriable. These different termination guarantees of activities will be defined more formally as follows using the notion of activity sequence to denote the sequential execution of activities.

Definition 1 (Effect-free Activities)
Let $\sigma =< a_i\ a_j \ldots a_n >$ *be a sequence of activities from* $\hat{\mathcal{A}}$. *The sequence* σ *is* effect-free *if, for all possible activity sequences* α *and* ω *from* $\hat{\mathcal{A}}$, *the return values of* α *and* ω *in the concatenated activity sequence* $< \alpha\ \sigma\ \omega >$ *are the same as in the activity sequence* $< \alpha\ \omega >$. □

A special case of effect-free activities is the sequence $\sigma =< a_i\ a_i^{-1} >$ consisting of a compensatable activity a_i and its compensating activity a_i^{-1}. More formally,

Definition 2 (Compensatability and Compensation)
An activity $a_i \in \hat{\mathcal{A}}$ *is* compensatable *if an activity* $a_i^{-1} \in \hat{\mathcal{A}}$ *exists where the activity sequence* $\sigma =< a_i\ a_i^{-1} >$ *is effect-free. The activity* a_i^{-1} *is then called the* compensating activity *of* a_i. □

In order to formally define retriable activities, the invocation of activities has to be labeled. Let $a_i(n)$ the n^{th} invocation of activity a_i.

Definition 3 (Retriable Activity)
An activity a_i *is* retriable *if some* $m \in \mathbf{N}$ *exists with* $a_i(j)$ *terminating with abort for* $1 \leq j < m$ *while* $a_i(m)$ *is guaranteed to terminate with commit.* □

The guarantee that there is always one invocation which will commit ensures that retriable activities will not fail. More formally,

Definition 4 (Failure of an Activity)
An activity a_i *has* failed *if invocation* $a_i(1)$ *has terminated with abort and no* $m \in \mathbf{N}$ *exists where* $a_i(m)$ *is guaranteed to commit.* □

To guarantee the property of compensatability, a compensating activity $a_{i_k}^{-1}$ is (i) itself not compensatable, however, it is (ii) retriable and therefore guaranteed to commit. Note further that according to the flex transaction model both pivot activities and retriable[2] activities do not have a compensating activity.

Intuitively, a process is an arbitrary collection of activities in arbitrary subsystems. For the process model, we adopt and refine ideas of the flex transaction model [ELLR90, ZNBB94]. More formally,

Definition 5 (Process)
A process, P, *is a triple* $(\mathcal{A}, \ll, \lhd)$, *where* $\mathcal{A} \subseteq \hat{\mathcal{A}}$ *is a set of activities,* $\ll$ *is a partial order over* $\mathcal{A}$ *with* $\ll \subseteq (\mathcal{A} \times \mathcal{A})$, *and* $\lhd$ *is a partial order defined over* $\ll$ *with* $\lhd \subseteq (\ll \times \ll)$ *establishing alternative execution paths by specifying for each activity* $a \in \mathcal{A}$ *an ordering on the activities* $a' \in \mathcal{A}$ *directly following it.* □

For notational purposes, a process is assumed to have a unique identifier, for instance, P_i. Activities within P_i are denoted as $a_{i_1}^c,\ a_{i_2}^p,\ \ldots,\ a_{i_n}^r$. The superscript index denotes the property of an activity, the subscript indices denote the process id and a unique id of the activity within the process (activity $a_{i_n}^r$, for instance, is an activity of process P_i with id n and it is retriable). The commitment of process P_i is denoted by C_i, its abort by A_i. If the property of an activity is not relevant, we will omit this specification.

The semantics of the precedence order $\ll$ within processes is a temporal one. This means that for any two activities, a_{i_k} and a_{i_l}, if $a_{i_k} \ll a_{i_l}$, then a_{i_l} can only be executed after a_{i_k} committed. The preference order $\lhd$ defined over pairs of connectors starting both from the same activity establishes the order in which the connectors will be evaluated. If there are two order constraints in $\ll$ with $(a_{i_h} \ll_{i_j} a_{i_j}) \lhd (a_{i_h} \ll_{i_k} a_{i_k})$ then, if a_{i_k} is executed, either a_{i_j} must have failed or both $a_{i_j}^c$ and $(a_{i_j}^c)^{-1}$ must have been executed. Also, all activities succeeding $a_{i_j}^c$ must have been compensated before a_{i_k} is able to be executed. Thus, as an extension of the flex transaction model, these further order constraints derived from $\ll$ have to be respected when executing alternatives. However, these alternative execution paths have the same semantics as the preference order of the flex transaction model. Note that both orders, $\ll$ and $\lhd$, are irreflexive, transitive, and acyclic. To avoid indeterminism in the execution, when, by transitivity, $\lhd$ associates several connectors, it can only define a total order.

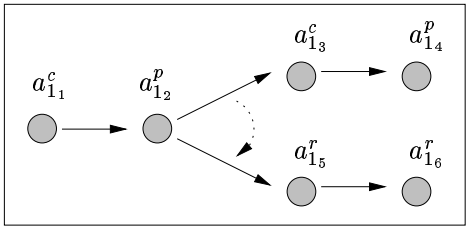

Figure 2: Process P_1 with precedence and preference order

[2] In the context of transactional process management, we could also consider retriable activities to be as well compensatable in order to give a scheduler more options for executing alternatives in case of failures. For the sake of simplicity, we however follow the less general flex transaction model here.

Example 1 *Consider process P_1 depicted in figure 2. The precedence order of P_1 is depicted with solid lines, the preference order of P_1 with dotted lines. Given these orders, $a^r_{1_5}$ and therefore also $a^r_{1_6}$ can only be executed after $a^c_{1_3}$ has failed or after $a^p_{1_4}$ has failed and $a^c_{1_3}$ has been compensated by $a^{-1}_{1_3}$. Therefore, as depicted in figure 3, four possible valid executions of P_1 exist.* □

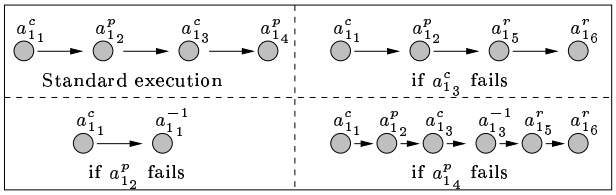

Figure 3: Possible executions of process P_1

We consider a single transactional process to be well defined if it has *well-formed flex structure* [ZNBB94]. The basic well-formed flex structure consists of a set of compensatable activities followed by one pivot activity which is again followed by a set of retriable activities. Additionally, the pivot activity can recursively be succeeded by a complete well-formed flex structure given that an alternative consisting only of retriable activities exists for it.

In [ZNBB94] it has been shown that well-formed flex structures always guarantee the existence of one execution path that can be executed correctly while all other paths will leave no effects. In the following, processes having well-formed flex structures are called *processes with guaranteed termination* (this is equivalent to the "semi atomicity" in the flex transaction model). The guaranteed termination property of transactional processes is a generalization of the "all-or-nothing" semantics of traditional ACID transactions as it ensures that at least one of eventually many valid executions (specified by the alternatives) is effected. In what follows, we will only consider processes with guaranteed termination.

For notational purposes, the first non-compensatable activity of a process with guaranteed termination P_i will be called *state-determining* activity s_{i_0} of P_i. All activities of P_i preceding s_{i_0} are compensatable. Therefore, backward recovery can be performed by successively applying compensation if s_{i_0} fails or if an abort A_i of P_i is performed before s_{i_0} committed. Similarly, once s_{i_0} has terminated successfully, forward recovery is guaranteed. From here, a process with guaranteed termination can be in any of two states. A process, P_i, is said to be *forward-recoverable*, $\mathcal{F}-\mathcal{REC}$, after s_{i_0} has been committed, otherwise P_i is *backward-recoverable*, $\mathcal{B}-\mathcal{REC}$. The sequence of compensating activities to be executed for recovery purposes of a process in state $\mathcal{B}-\mathcal{REC}$ is its *backward recovery path*. The sequence of activities leading from any activity succeeding s_{i_0} to the well-defined termination of a process is the *forward recovery path*. The set of activities of a process P_i to be executed for recovery purposes (either forward or backward) will be called the *completion* of P_i denoted by $\mathcal{C}(P_i)$. Note that in the case of P_i being in state $\mathcal{B}-\mathcal{REC}$, $\mathcal{C}(P_i)$ consists only of compensating activities, while, if P_i is in state $\mathcal{F}-\mathcal{REC}$, $\mathcal{C}(P_i)$ consists of both compensating activities (local backward recovery to a state-determining element s_{i_k})[3], and retriable activities. While the failure of one activity leads to the execution of the next alternative given by the preference order $\lhd$, the abort A_i of a process in $\mathcal{F}-\mathcal{REC}$ considers only the alternative with lowest priority which consists only of retriable activities and thus guarantees safe termination. Similarly, the abort A_i of a process P_i in $\mathcal{B}-\mathcal{REC}$ considers only compensation in backward order and no further alternative execution paths. The completion $\mathcal{C}(P_i)$ of a process P_i will be an important notion when we define complete process schedules below.

[3] As we consider basic well-formed flex structures recursively, multiple local state-determining activities s_{i_k} of P_i may exist.

Example 2 *Consider again process P_1 depicted in figure 2. Obviously, P_1 is a process with guaranteed termination as it has well-formed flex structure. The pivot activity $a^p_{1_2}$ is the state-determining activity s_{1_0} of P_1. Before the successful termination of $a^p_{1_2}$, P_1 is in $\mathcal{B}-\mathcal{REC}$ and in this state, the completion $\mathcal{C}(P_1)$ consists of $\{a^{-1}_{1_1}\}$ if $a^c_{1_1}$ has been executed correctly. After successful termination of $a^p_{1_2}$, P_1 is in $\mathcal{F}-\mathcal{REC}$. After activity $a^c_{1_3}$, for instance, has terminated successfully, the completion of P_1 evaluates to $\mathcal{C}(P_1) = \{a^{-1}_{1_3} \ll a^r_{1_5} \ll a^r_{1_6}\}$.* □

3.2 Process Schedules and Correctness

Following [VHYBS98], the notion of conflicting activities is defined using the return values of activities.

Definition 6 (Commutativity)
Two activities $a_{i_k}, a_{j_l} \in \hat{\mathcal{A}}$ commute *if for all activity sequences α and ω from $\hat{\mathcal{A}}$, the return values in the concatenated activity sequence $< \alpha\ a_{i_k}\ a_{j_l}\ \omega >$ are identical to the return values of the activity sequence $< \alpha\ a_{j_l}\ a_{i_k}\ \omega >$.* □

Two activities are in conflict if they do not commute. Furthermore, we consider commutativity between all activities of $\hat{\mathcal{A}}$ to be perfect [VHYBS98]. This means that if two activities $a^c_{i_k}$ and a_{j_l} conflict, then we will also consider a conflict between $a^\alpha_{i_k}$ and $a^\beta_{j_l}$ for all possible combinations of $\alpha, \beta \in \{-1, 1\}$. Otherwise, if $a^c_{i_k}$ and a_{j_l} commute, we will assume $a^\alpha_{i_k}$ and $a^\beta_{j_l}$ to commute for all possible combinations of $\alpha, \beta \in \{-1, 1\}$.

Given the structure of processes with guaranteed termination and the information about conflicting activities, a process schedule can be defined as follows.

Definition 7 (Process Schedule)
A process schedule *S is a triple $(\mathcal{P}_S, \mathcal{A}_S, \ll_S)$ where $\mathcal{P}_S$ is a set of processes, $\mathcal{A}_S \subseteq \hat{\mathcal{A}}$ is a subset of all activities of all processes of $\mathcal{P}_S$ with $\mathcal{A}_S \subseteq \{a_{i_j} |\ a_{i_j} \in \mathcal{A}_i \wedge P_i \in \mathcal{P}_S\}$, $\ll_S$ is a partial order between activities of $\mathcal{A}_S$ with $\ll_S \subseteq (\mathcal{A}_S \times \mathcal{A}_S)$. For the order $\ll_s$ the following has to hold:*

1. $\forall\ P_i : \ll_i \subseteq \ll_S$
2. $\forall\ (a_{i_k}, a_{j_l}), i \neq j$, *such that a_{i_k} and a_{j_l} do not commute: $a_{i_k} \ll_S a_{j_l}$ or $a_{j_l} \ll_S a_{i_k}$* □

Note that by 7.1, a process schedule guarantees only legal executions of each process $P_i \in \mathcal{P}_S$ thus respecting both P_i's precedence and preference order.

Formally, the above definition of a process schedule looks like the classical definition of a schedule. However, it implicitly includes information about the properties of all activities (compensatable, pivot or retriable) and thus, also about the different states of processes ($\mathcal{B}-\mathcal{REC}$ or $\mathcal{F}-\mathcal{REC}$) and it includes the alternative execution of a process P_i as even

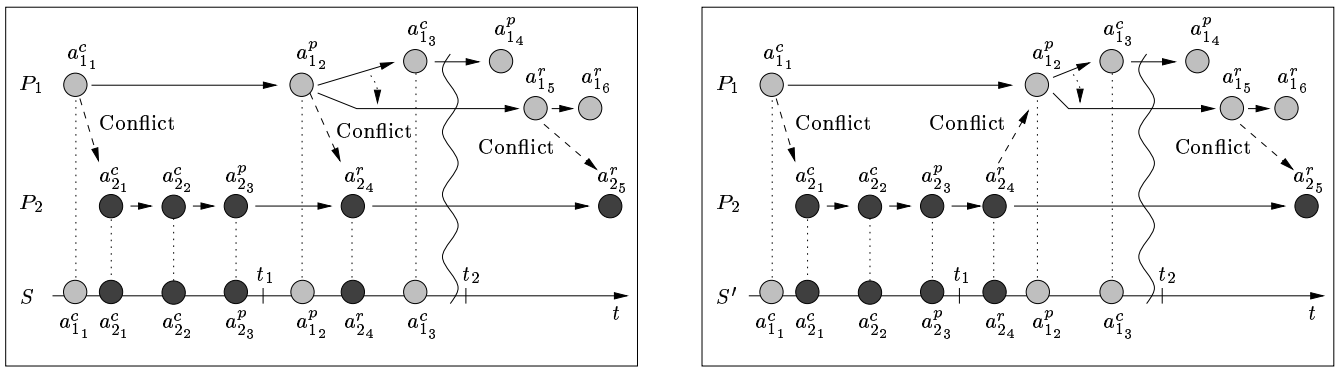

Figure 4: Serializable (a) and non-serializable (b) concurrent execution of processes P_1 and P_2

in a complete process schedule where all processes terminate committing [BHG87], not necessarily all of P_i's activities are considered. This does however not influence the notion of serializability. A process schedule is serializable if it is conflict equivalent to a serial execution of all processes. Hence, a serializable process schedule does not contain cyclic dependencies [BHG87].

Example 3 *Consider the two processes, P_1 and P_2, depicted in figure 4(b) being executed in parallel. As the pairs of activities $(a^c_{1_1}, a^c_{2_1})$, $(a^p_{1_2}, a^r_{2_4})$, and $(a^r_{1_5}, a^r_{2_5})$ do not commute (denoted by dashed arcs), they have to be ordered in the process schedule S'. Also, the intra-process orders of P_1 and P_2 must be respected in S'. Therefore, process schedule S' at time t_2 evaluates to: $S'_{t_2} = (\mathcal{P}_{S'_{t_2}}, \mathcal{A}_{S'_{t_2}}, \ll_{S'_{t_2}})$ with the set of processes $\mathcal{P}_{S'_{t_2}} = \{P_1, P_2\}$, the set of activities $\mathcal{A}_{S'_{t_2}} = \{a^c_{1_1}, a^p_{1_2}, a^c_{1_3}, a^c_{2_1}, a^c_{2_2}, a^p_{2_3}, a^r_{2_4}\}$, and the order $\ll_{S'_{t_2}} = \{(a^c_{1_1} \ll_{S'_{t_2}} a^p_{1_2} \ll_{S'_{t_2}} a^c_{1_3}), (a^c_{2_1} \ll_{S'_{t_2}} a^c_{2_2} \ll_{S'_{t_2}} a^p_{2_3} \ll_{S'_{t_2}} a^r_{2_4}), (a^c_{1_1} \ll_{S'_{t_2}} a^c_{2_1}), (a^r_{2_4} \ll_{S'_{t_2}} a^p_{1_2})\}$. Obviously, process schedule S'_{t_2} is not serializable because of cyclic dependencies between P_1 and P_2.* □

Example 4 *Consider again processes P_1 and P_2, now executed as depicted in figure 4(a). At time t_2, the process schedule S_{t_2} is serializable. Here, no cyclic dependencies between P_1 and P_2 do exist as the order $\ll_{S_{t_2}}$ evaluates to $\ll_{S_{t_2}} = \{(a^c_{1_1} \ll_{S_{t_2}} a^p_{1_2} \ll_{S_{t_2}} a^c_{1_3}), (a^c_{2_1} \ll_{S_{t_2}} a^c_{2_2} \ll_{S_{t_2}} a^p_{2_3} \ll_{S_{t_2}} a^r_{2_4}), (a^c_{1_1} \ll_{S_{t_2}} a^c_{2_1}), (a^p_{1_2} \ll_{S_{t_2}} a^r_{2_4})\}$.* □

3.3 Completed Process Schedules

The serializability of transactional processes allows to reason about correct concurrency control. In order to additionally reason about correct recovery when, for instance, a failure of the process scheduler occurs, we now make recovery-related activities explicit by applying the unified theory of concurrency control and recovery [SWY93, AVA+94, VHYBS98] to transactional processes. Therefore, we replace each abort activity A_i of a process P_i by the activities of its completion $\mathcal{C}(P_i)$. This replacement of abort activities leads to the notion of the *completed process schedule* $\tilde{S}$. In order to guarantee correct recovery, all active processes $P_{i_1}, \ldots, P_{i_n}$ are assumed to abort, which must be treated jointly by using a group abort operation $A(P_{i_1}, \ldots, P_{i_n})$. Note that aborted processes may be in $\mathcal{F} - \mathcal{REC}$. Therefore, not only compensation of previously executed activities but all activities of the forward-recovery path of aborted processes have to be considered, thus leading to crucial differences compared with the standard undo procedure for recovery. This is also reflected in the notion of completed process schedule in contrast to the expanded schedule of the traditional unified theory which contains only additional compensation compared with the initial schedule. The way a process schedule is completed is depicted in figure 5. After A_i has been replaced by all activities of $\mathcal{C}(P_i)$, a process P_i can be considered as committed.

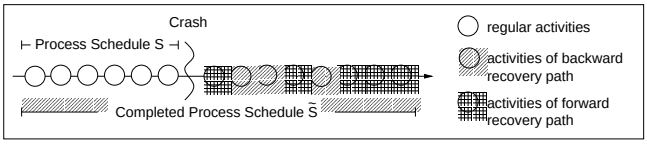

Figure 5: Completion of a process schedule by activities of the backward recovery path and of the forward recovery path of all active processes

More formally, the completed process schedule $\tilde{S}$ of a process schedule S is defined as follows:

Definition 8 (Completed Process Schedule $\tilde{S}$) *Let $S = (\mathcal{P}_S, \mathcal{A}_S, \ll_S)$ be a process schedule. The* completed process schedule $\tilde{S}$ *of S, is a triple $(\tilde{\mathcal{P}}_S, \tilde{\mathcal{A}}_S, \tilde{\ll}_S)$ where*

1. *For the set of processes $\tilde{\mathcal{P}}_S$ holds: $\tilde{\mathcal{P}}_S = \mathcal{P}_S$.*
2. *$\tilde{\mathcal{A}}_S$ is a set of activities derived from $\mathcal{A}_S$ in the following way:*
   (a) *For each process $P_i \in \mathcal{P}_S$, if $a_{i_k} \in \mathcal{A}_i$ and a_{i_k} is not the abort activity A_i, then $a_{i_k} \in \tilde{\mathcal{A}}_S$.*
   (b) *All active processes are treated as aborted processes, by adding $A(P_{n_1}, \ldots, P_{n_s})$, a set-oriented abort, at the end of S, where $(P_{n_1}, \ldots, P_{n_s})$ are all active processes in S.*
   (c) *For each aborted process P_j in $\mathcal{P}_S$, all activities $a_{j_s} \in \mathcal{C}(P_j)$ of the completion $\mathcal{C}(P_j)$ of P_j are in $\tilde{S}$ ($a_{j_s} \in \tilde{\mathcal{A}}_S$). An abort activity A_j is changed to $C_j \in \tilde{\mathcal{A}}_S$.*

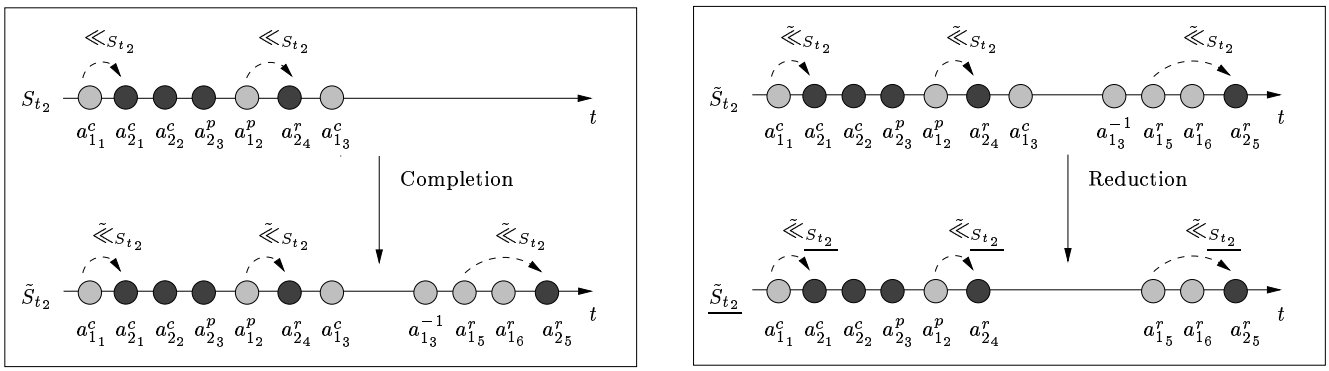

Figure 6: Completed process schedule $\tilde{S}_{t_2}$ (a) and reduced process schedule $\underline{\tilde{S}_{t_2}}$ (b) of process schedule S_{t_2}

3. *The partial order, $\tilde{\ll}_S$, is determined as follows:*

   (a) *For every two activities, a_{i_k} and a_{j_l}, if $a_{i_k} \ll_S a_{j_l}$ in S, then $a_{i_k} \tilde{\ll}_S a_{j_l}$ in $\tilde{S}$.*

   (b) *For every two activities, a_{i_k} and a_{i_l}, of the completion $\mathcal{C}(P_i)$ of every process P_i that does not commit in S, if $a_{i_k} \ll_i a_{i_l} \in \mathcal{C}(P_i)$, then $a_{i_k} \tilde{\ll}_S a_{i_l}$ in $\tilde{S}$.*

   (c) *All activities of the completion $\mathcal{C}(P_i)$ of every process P_i that does not commit in S follow the P_i original activities and must precede C_i in $\tilde{S}$.*

   (d) *If a group abort $A(P_{n_1}, \ldots, P_{n_s}) \in S$, then every pair of conflicting activities of the completions of these processes, $a_{i_k} \in \mathcal{C}(P_i), a_{j_l} \in \mathcal{C}(P_j)$ with $i, j \in \{n_1, \ldots, n_s\}, i \neq j$, has to be ordered in $\tilde{S}$ (either $a_{i_k} \tilde{\ll}_S a_{j_l}$ or $a_{j_l} \tilde{\ll}_S a_{i_k}$).*

   (e) *Whenever $a_{i_k} \ll_S A(P_{n_1}, \ldots, P_{n_s}) \ll_S a_{j_l}$ and some activity a_{q_t} of the completion $\mathcal{C}(P_q)$ of process $P_q \in \{P_{n_1}, \ldots, P_{n_s}\} \subseteq \mathcal{P}_S$ conflicts with a_{j_l} (a_{i_k}), then it must be true that $a_{q_t} \tilde{\ll}_S a_{j_l}$ ($a_{i_k} \tilde{\ll}_S a_{q_t}$).*

   (f) *Whenever $A(\ldots, P_i, \ldots) \ll_S A(\ldots, P_j, \ldots)$ for some $i \neq j$, then for all conflicting activities a_{i_k} of the completion of P_i and a_{j_l} of the completion of P_j, $a_{i_k} \in \mathcal{C}(P_i)$ and $a_{j_l} \in \mathcal{C}(P_j)$, it must be true that $a_{i_k} \tilde{\ll}_S a_{j_l}$.* □

The following example presents how a given process schedule is completed.

Example 5 *Consider again process schedule S_{t_2} of example 4 with $\mathcal{P}_{S_{t_2}} = \{P_1, P_2\}$ as depicted in figure 4(a). When the completed process schedule $\tilde{S}$ is determinded at time t_2 where both processes are active, a group abort $A(P_1, P_2)$ has to be added to S_{t_2}. The set of activities $\tilde{\mathcal{A}}_{S_{t_2}}$ of $\tilde{S}_{t_2}$ consists of all activities of $\mathcal{A}_{S_{t_2}}$ plus the activities $\{a_{1_3}^{-1}, a_{1_5}^r, a_{1_6}^r\}$ of the completion $\mathcal{C}(P_1)$ and $\{a_{2_5}^r\}$ of the completion $\mathcal{C}(P_2)$. The order $\tilde{\ll}_{S_{t_2}}$ of $\tilde{S}_{t_2}$ is the union of $\ll_{S_{t_2}}$ and $\{(a_{1_3}^c \tilde{\ll}_{S_{t_2}} a_{1_3}^{-1} \tilde{\ll}_{S_{t_2}} a_{1_5}^r \tilde{\ll}_{S_{t_2}} a_{1_6}^r), (a_{2_4}^r \tilde{\ll}_{S_{t_2}} a_{2_5}^r), (a_{1_5}^r \tilde{\ll}_{S_{t_2}} a_{2_5}^r)\}$. The completed process schedule $\tilde{S}_{t_2}$ is depicted in figure 6(a). As no cyclic dependencies exist, the completed process schedule $\tilde{S}_{t_2}$ is serializable.* □

3.4 Unified Theory for Processes

Like in the traditional unified theory, *reducibility* provides a criterion for correct concurrency control and recovery once we have completed a process schedule by making recovery-related activities explicit. The idea of the reduction of a completed process schedule is to eliminate both an activity and its compensating activity if they form an effect-free activity sequence as well as to eliminate activities of aborted processes that are themselves effect-free. Also, consecutive activities may be commuted if they do not conflict. More formally,

Definition 9 (Reducibility (RED))
A process schedule $S = (\mathcal{P}_S, \mathcal{A}_S, \ll_S)$ is reducible (RED) *if its completed process schedule $\tilde{S} = (\tilde{\mathcal{P}}_S, \tilde{\mathcal{A}}_S, \tilde{\ll}_S)$ can be transformed into a serial process schedule $\underline{\tilde{S}} = (\tilde{\mathcal{P}}_{\underline{S}}, \tilde{\mathcal{A}}_{\underline{S}}, \tilde{\ll}_{\underline{S}})$ by applying the following three transformation rules finitely many times:*

1. **Commutativity Rule:** *If two activities $a_{i_k}, a_{j_l} \in \tilde{\mathcal{A}}_S$ such that $a_{i_k} \tilde{\ll}_S a_{j_l}$ and (a_{i_k}, a_{j_l}) commute and there is no other activity $a_{q_t} \in \tilde{\mathcal{A}}_S$ with $a_{i_k} \tilde{\ll}_S a_{q_t} \tilde{\ll}_S a_{j_l}$, then the ordering $a_{i_k} \tilde{\ll}_S a_{j_l}$ can be replaced by the ordering $a_{j_l} \tilde{\ll}_{\underline{S}} a_{i_k}$.*

2. **Compensation Rule:** *If two activities $a_{i_k}, a_{i_k}^{-1} \in \tilde{\mathcal{A}}_S$ such that $a_{i_k} \tilde{\ll}_S a_{i_k}^{-1}$ and there is no other activity $a_{j_l} \in \tilde{\mathcal{A}}_S$ with $a_{i_k} \tilde{\ll}_S a_{j_l} \tilde{\ll}_S a_{i_k}^{-1}$, then $a_{i_k}, a_{i_k}^{-1}$ can be removed from $\underline{\tilde{S}}$.*

3. **Effect-free Activity Rule:** *If P_i does not commit in S, then all activities a_{i_k} that are effect-free can be removed from $\underline{\tilde{S}}$.* □

Example 6 *Considering again process schedule S_{t_2} of example 4 and its completed process schedule $\tilde{S}_{t_2}$ of example 5. When applying the reduction rules, only the two consecutive activities $a_{1_3}^c$ and $a_{1_3}^{-1}$ can be removed from $\tilde{S}_{t_2}$ in accordance to the compensation rule. The reduced process schedule $\underline{\tilde{S}_{t_2}}$ shown in figure 6(b) is serializable as $\tilde{\ll}_{\underline{S_{t_2}}}$ of $\underline{\tilde{S}_{t_2}}$ contains aside of the inter-process orders of P_1 and P_2 only dependencies from process P_1 to process P_2. Therefore, process schedule S_{t_2} is RED.* □

Example 7 *Consider now process schedule S''_{t_1} at time t_1 depicted in figure 7. When completing S''_{t_1}, all pairs of conflicting activities will be in the same order and the application of the reduction rules leads to a serial process schedule $\tilde{S}''_{t_1}$. Therefore, process schedule S''_{t_1} is RED.* □

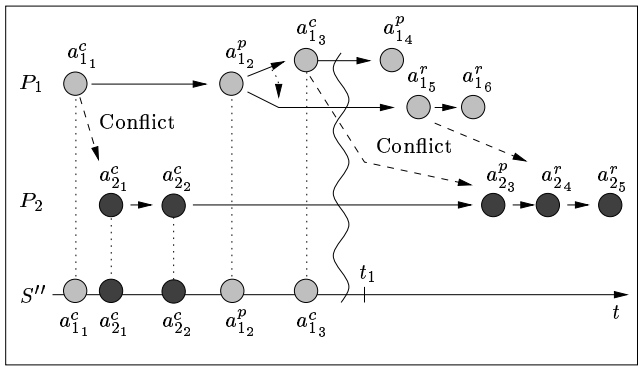

Figure 7: Prefix-reducible execution of processes P_1 and P_2

RED is not prefix closed, which means, it cannot be used for dynamic scheduling. In accordance to the traditional unified theory, the criterion can be further restricted for this purpose leading to *prefix-reducibility* where each prefix of a process schedule has to be considered. More formally,

Definition 10 (Prefix-Reducibility (PRED)) *A process schedule $S = (\mathcal{P}_S, \mathcal{A}_S, \ll_S)$ is* prefix-reducible (PRED) *if every prefix of S is reducible.* □

Example 8 *Consider again process schedule S_{t_2} of example 4 depicted in figure 4(a) and its prefix S_{t_1} at time t_1. In S_{t_1}, process P_2 is in $\mathcal{F}-\mathcal{REC}$ while process P_1 is in $\mathcal{B}-\mathcal{REC}$. When completing S_{t_1}, the previously executed activity $a^c_{1_1}$ of P_1 has to be compensated by $a^{-1}_{1_1}$ while for P_2, the activities of the forward recovery path have to be executed. By scheduling $a^{-1}_{1_1}$, a conflict cycle appears in $\tilde{S}_{t_1}$ ($a^c_{1_1} \ll_{\tilde{S}_{t_1}} a^c_{2_1} \ll_{\tilde{S}_{t_1}} a^{-1}_{1_1}$) that cannot be eliminated by the reduction rules as compensation of $a^c_{2_1}$ is not available. Therefore, S_{t_1} is not reducible and thus, S_{t_2} is not prefix-reducible. The completed process schedule $\tilde{S}_{t_1}$ of S_{t_1} is depicted in figure 8.* □

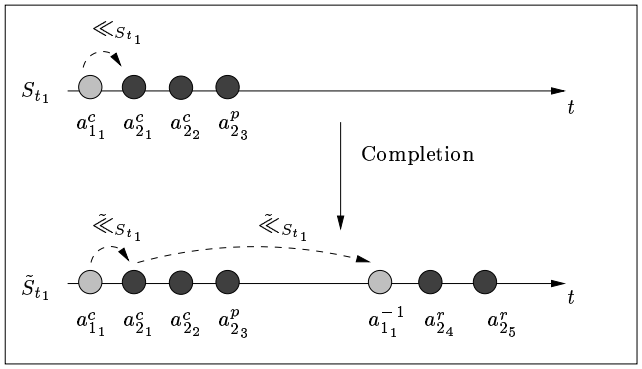

Figure 8: Completed process schedule $\tilde{S}_{t_1}$ of S_{t_1}

Note that the above example is strongly influenced by the fact that activities without inverse do exist. Therefore, we have to consider not only compensation for recovery purposes. If all inverses were available and the classical undo procedure of recovery could be applied, the prefix S_{t_1} of S_{t_2} would be reducible. The completion of S_{t_1} would consider the compensation of $a_{2_3}, a_{2_2}, a_{2_1}$, and a_{1_1}. Then, with respect to the compensation rule, all four activities and their compensation activity could be removed from $\tilde{S}_{t_1}$ leading to a reduced schedule $\underline{\tilde{S}_{t_1}}$ consisting only of C_1 and C_2. As reduction would be possible for all prefixes of S_{t_2} in this classical sense, S_{t_2} would be in PRED. Therefore, when considering transactional processes with guaranteed termination property, the order in which non-compensatable activities are executed is crucial as we will see in section 3.5.

Example 9 *Taking again a look at process schedule S''_{t_1} depicted in figure 7. It can be shown that each prefix $S''_{t'}$ of S''_{t_1} with $t' < t_1$ is reducible. Therefore, process schedule S''_{t_1} is PRED.* □

However, scheduling can also benefit from non-compensatable activities. They have the semantics of a "quasi commit" of a process, as for all activities $a^c_{i_k}$ of a process P_i preceding such a non-compensatable activity s_i, compensation can no longer be considered. Therefore, after the commitment of s_i, no cyclic conflicts can arise in the completed process schedule by the compensation activities $a^{-1}_{i_k}$. This is shown in the following example.

Example 10 *Consider process schedule S^* with processes P_1 and P_3 depicted in figure 9. Although activities $a^c_{1_1}$ and $a^c_{3_1}$ do conflict, no conflict cycle can appear by the compensating activity $a^{-1}_{1_1}$ at time t_1. As process process P_1 is already in $\mathcal{F}-\mathcal{REC}$, compensation of $a^c_{1_1}$ is not available. Therefore, given that no further conflicts exist between activities of P_3 and the activities of the forward recovery path of P_1, the execution depicted in figure 9 is correct with respect to both concurrency control and recovery.* □

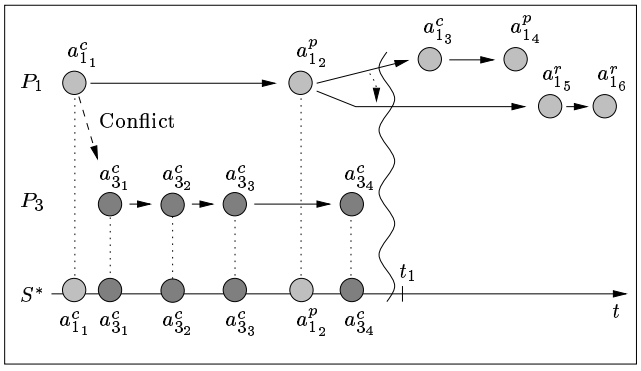

Figure 9: Correct interleaving of processes exploiting the "quasi-commit" of non-compensatable activities

3.5 Discussion of PRED of Completed Process Schedules

In the previous sections, we introduced the formalism needed to define prefix-reducibility with respect to transactional processes having guaranteed termination property. As our goal is to reason about correct concurrency control and recovery, we have to prove that each process schedule in PRED

is in fact both serializable and recoverable. As we have to deal with two different states of processes determining the way recovery has to be performed, we have to adopt the notion of recoverability to the structure of transactional processes leading to the notion of *process-recoverability*. More formally,

Definition 11 (Process-Recoverability (Proc-REC)) *A process schedule S is* process-recoverable (Proc-REC), *if for each pair of conflicting activities, a_{i_k} and a_{j_l} with $a_{i_k} \ll_S a_{j_l} \in S$ the following holds:*

1. *C_i precedes C_j in S ($C_i \ll_S C_j$)*
2. *the next non-compensatable activity a_{j_m} of P_j following a_{j_l} succeeds in S the next next non-compensatable activity a_{i_n} of P_i following a_{i_k} ($a_{i_n} \ll_S a_{j_m}$).* □

Note that in the above definition, the traditional case where no non-compensatable activities exist is contained as then, by definition 11.1, only an order between C_i and C_j with $C_i \ll_S C_j$ has to be imposed.

Theorem 1 *If a process schedule S is PRED, then S is both serializable and process-recoverable.* □

The proof of theorem 1 is given in appendix A.

In example 8, we have seen that the order in which the state-determining elements of conflicting processes are executed is crucial as it determines what is to be done in case of recovery (either forward or backward). We now formalize and generalize this dependency.

Lemma 1 *For each process schedule S in PRED with two conflicting activities $a_{i_k} \ll_S a_{j_l}$ in S where process P_i is active, the following has to hold:*

1. *Each non-compensatable activity a_{j_m} of P_j with $a_{j_l} \ll_j a_{j_m}$ has to succeed the commit C_i of P_i ($C_i \ll_S a_{j_m}$).*
2. *Activity a_{j_l} has to be compensatable ($a^c_{j_l}$).* □

The proof of lemma 1 is given in appendix B.

In schedule S_{t_1} of example 8 with the pair of conflicting activities ($a^c_{1_1} \ll_{S_{t_1}} a^c_{2_1}$), $a^p_{2_2}$ is executed before $a^p_{1_2}$ and thus, P_2 is in $\mathcal{F}-\mathcal{REC}$ while process P_1 is still in $\mathcal{B}-\mathcal{REC}$ leading to a contradiction of lemma 1.1 and a violation of the PRED criterion.

According to lemma 1, the commits of all non-compensatable activities of P_j have to be deferred by the respective subsystem until process P_i has committed (C_i) if a conflict between some activity a_{i_k} and a_{j_l} with $a_{i_k} \ll_S a_{j_l}$ exists in S. After P_i has committed, all non-compensatable activities of P_j are also allowed to commit as cyclic dependencies between P_i and P_j can no longer appear. Thus, the commitment of all non-compensatable activities of P_j has to be performed atomically by exploiting a two phase commit protocol in order to ensure that either all activities commit or none of them.

In the following, we analyze the implications, PRED has on the execution of activities within the completed process schedule. The following two lemmas specify the restrictions on the execution of compensating activities.

Intuitively, all compensating activities have to be in reverse order of the original activities. More formally:

Lemma 2 *For each process schedule S in PRED with two conflicting activities $a^c_{i_k}$ and $a^c_{j_l}$, if both compensating activities $a^{-1}_{i_k}$ and $a^{-1}_{j_l}$ are in the completed process schedule $\tilde{S}$, then they have to be in reverse order of the two corresponding activities in S.* □

The proof of lemma 2 is given in appendix C.

As we have to consider not only compensating activities for recovery purposes, additional restrictions between compensating activities of $\mathcal{C}(P_i)$ for some P_i in state $\mathcal{B}-\mathcal{REC}$ and non-compensatable activities ($a^r_{j_l}$) of $\mathcal{C}(P_j)$ for some P_j in state $\mathcal{F}-\mathcal{REC}$ have to be considered.

Lemma 3 *For each process schedule S in PRED, if two conflicting activities $a^{-1}_{i_k} \in \mathcal{C}(P_i)$ and a non-compensatable activity $a^r_{j_l} \in \mathcal{C}(P_j)$ have to be executed when completing S, then $a^{-1}_{i_k}$ has to precede $a^r_{j_l}$ in $\tilde{S}$ ($a^{-1}_{i_k} \ll_{\tilde{S}} a^r_{j_l}$).* □

The proof of lemma 3 is given in appendix D.

Coming back to the initial CIM example presented in section 2, we now have a formal criterion to classify the execution depicted in figure 1 as incorrect because the PRED criterion does not hold. In order to guarantee correctness, the production activity would have to be deferred until the commitment of the construction process.

Unlike the traditional unified theory where only compensation had to be considered for aborted transactions in the expanded schedule, here also new activities have to be scheduled when the completed process schedule has to be built. Thus, aside from already existing pairs of conflicting processes (if some undo operation is in conflict with an activity of another transaction in the traditional model, a conflict between both transactions must have been existed before compensation has been performed), new conflicts between processes may be introduced. Therefore, unlike in the traditional unified theory, the completed process schedule $\tilde{S}$ has always to be considered when reasoning about correctness of a process schedule for transactional processes.

In [AVA+94], the criterion SOT (serializable with ordered termination) has been introduced in order to reason about correct concurrency control and recovery of a schedule S without considering its expanded schedule $\tilde{S}$. However, as the activities of the completion of a process are not known in advance, a SOT-like criterion (that relies only on information of a given schedule S) does not exist for transactional processes. Arbitrary conflicts can be introduced to $\tilde{S}$ when non-compensatable activities of $\mathcal{C}(P_i)$ of aborted processes P_i have to be considered. Therefore, when reasoning about correct concurrency control and recovery of transactional processes, the completed process schedule $\tilde{S}$ has always to be considered to evaluate the PRED criterion.

3.6 Increasing Parallelism of Conflicting Activities

In the process model (definition 5), we only allowed either sequential execution ($\ll$) of activities or unrestricted parallelism. Also, in definition 7 of a process schedule, we only considered a (strong) temporal order ($\ll_S$) between two conflicting activities. In order to increase parallelism, the weak order taken from the composite systems theory [ABFS97] could be applied with respect to the hierarchical schedulers of the type encountered when executing transactional processes on top of transactional subsystems. In this configuration, the output of the process scheduler is used as input to

several lower schedulers, the schedulers of the transactional subsystems. Thus, this reflects the case of fork schedules described in [AFPS99]. While the strong order enforces sequential execution, i.e., an activity is invoked only after the previous one has terminated, the weak order between two activities is more permissive, meaning that both activities can be executed in parallel as long as the overall effect is the same as if they would have been executed as specified by the strong order. The differentiation between strong and weak order can be made both within processes (intra-process order) and within conflicting activities of different processes (inter-process order). Then, all pairs of conflicting activities have to be weakly ordered as indicated by the composite transaction model. The subsystem is then responsible for keeping this weak order when executing both conflicting activities in parallel. In order to ensure this weak order, a subsystem has, for instance, to provide a protocol supporting commit order serializability [BBG89]. Then, the commit order can be derived from the weak order between conflicting activities. Otherwise (if the weak order is not supported by the subsystem), as the weak order always contains the strong one, conflicting activities have to be executed with respect to a strong order.

The re-invocation of retriable activities now may lead to a special treatment of other activities executed in parallel. Suppose two activities $a^r_{i_k}$ and a_{j_l}, with $a^r_{i_k} <_S a_{j_l}$, have to be executed within the same subsystem. If the local transaction T_{i_k} corresponding to $a^r_{i_k}$ terminates aborting after some operations of T_{i_k} have already been executed, then, in general, the local transaction T_{j_l} (which corresponds to activity a_{j_l}) running in parallel to T_{i_k} (with respect to the given weak order) has to be aborted, too. However, as this is not due to a failure of T_{j_l}, it must not lead to an exception of P_j leading to an other alternative. Moreover, after T_{i_k} is restarted, T_{j_l} has to be restarted within the subsystem, too, hence guaranteeing compliance to the weak order between both transactions.

The integration of the composite systems ideas into the process model and the process schedule are described in detail in [SAS99].

4 Conclusion

This paper provides a framework to jointly reason about correct concurrency control and recovery for transactional processes in order to ensure both a more general notion of atomicity (guaranteed termination) by the flexible handling of failures with appropriate alternative executions and correct interleavings of parallel processes. Unlike other approaches addressing only parts of this problem, we cover both atomicity and isolation simultaneously and do concurrency control and recovery at the appropriate level, the scheduling of processes. Furthermore, with the theory of composite systems, we can take into account the interaction between hierarchical schedulers when executing transactional processes and increase parallelism by treating them according to the weak conflict order.

With PRED, we have provided a correctness criterion for transactional processes based on the notion of completed process schedules. We have additionally shown that, due to the structure of transactional processes, the SOT correctness criterion cannot be applied. Because of the execution of non-compensatable activities during the completion of a process, reasoning about process recovery becomes more complex than in the traditional case where only compensation has to be applied. Therefore, the completed process schedule has to be considered. Furthermore, we have identified important prerequisites of PRED schedules that have to be respected due to the fact that some activities might be non-compensatable. Therefore, aside of the atomicity of single activities and the compliance of orderings, the deferred commit of all non-compensatable activities and their atomic commit by exploiting a two phase commit protocol has to be provided by the subsystems.

The framework established in this paper not only covers various applications such as workflow management, process support systems, and the provision of appropriate infrastructures for electronic commerce, virtual enterprises, and the CIM scenario presented in section 2, it is also completely transparent to the user. Within the WISE project of ETH Zürich [AFH+99], we have implemented a process scheduler for transactional process management using a protocol which is based on the correctness criterion presented in this paper. This complements the correctness checking of single processes with respect to their guaranteed termination property which is also available within the WISE system. The two ideas complete the effort to provide execution guarantees for transactional processes. Based on them, we will in our future work expand the framework established in this paper to identify transactional execution guarantees of subprocesses and to reason about decoupled execution guarantees of subprocesses.

References

[AAHD97] I. Arpinar, S. Arpinar, U. Halici, and A. Dogac. Correctness of Workflows in the Presence of Concurrency. In *Proceedings of the Next Generation Information Technologies and Systems Conference (NGITS'97)*, Israel, June 1997.

[ABFS97] G. Alonso, S. Blott, A. Feßler, and H.-J. Schek. Correctness and Parallelism in Composite Systems. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'97)*, Tucson, Arizona, May 12-15 1997.

[AFH+99] G. Alonso, U. Fiedler, C. Hagen, A. Lazcano, H. Schuldt, and N. Weiler. WISE: Business to Business E-Commerce. In *Proceedings of the 9th International Workshop on Research Issues on Data Engineering. Information Technology for Virtual Enterprises (RIDE-VE'99)*, Sydney, Australia, March 1999.

[AFPS99] G. Alonso, A. Feßler, G. Pardon, and H.-J. Schek. Transactions in Stack, Fork and Join Composite Systems. In *Proceedings of the 7th International Conference on Database Theory (ICDT'99)*, Jerusalem, Israel, January 1999.

[Alo97] G. Alonso. Processes + Transactions = Distributed Applications. In *Proceedings of the High Performance Transaction Processing Workshop (HPTS'97)*, Asilomar, California, September 1997.

[AVA+94] G. Alonso, R. Vingralek, D. Agrawal, Y. Breitbart, A. El Abbadi, H.-J. Schek, and G. Weikum. Unifying Concurrency Control and Recovery of Transactions. *Information Systems*, 19(1):101–115, 1994.

[BBG89] C. Beeri, P.A. Bernstein, and N. Goodman. A model for concurrency in nested transaction systems. *Journal of the Association for Computing Machinery*, 36(2):230–269, April 1989.

[BHG87] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

[ELLR90] A. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A Multidatabse Transaction Model for InterBase. In *Proceedings of the* 16^{th} *VLDB Conference*, pages 507–518, Brisbane, Australia, 1990.

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

[GHKM94] D. Georgakopoulos, M. Hornik, P. Krychniak, and F. Manola. Specification and Management of Extended Transactions in a Programmable Transaction Environment. In *Proceeding of the* 10^{th} *International Conference on Data Engineering (ICDE'94)*, pages 462–473, Houston, Texas, February 1994.

[GHS95] D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.

[JK97] S. Jajodia and L. Kerschberg, editors. *Advanced Transaction Models and Architectures.* Kluwer Academic Publishers, 1997.

[KR98] M. Kamath and K. Ramamritham. Failure Handling and Coordinated Execution of Concurrent Workflows. In *Proceedings of the* 14^{th} *International Conference on Data Engineering (ICDE'98)*, pages 334–341, Orlando, Florida, February 1998.

[Ley95] F. Leymann. Supporting Business Transactions via Partial Backward Recovery in Workflow Management Systems. In *Datenbanksysteme in Büro, Technik und Wissenschaft*, pages 51–70, 1995.

[MRSK92] S. Mehrotra, R. Rastogi, A. Silberschatz, and H. Korth. A Transaction Model for Multidatabase Systems. In *Proceedigs of the* 12^{th} *International Conference on Distributed Computing Systems (ICDCS'92)*, pages 56–63, Yokohama, Japan, June 1992.

[NSSW94] M. Norrie, W. Schaad, H.-J. Schek, and M. Wunderli. CIM Through Database Coordination. In *Proceedings of the International Conference on Data and Knowledge Systems*, May 1994.

[RSS97] A. Reuter, K. Schneider, and F. Schwenkreis. *ConTracts Revisited*, chapter 5. In: [JK97]. Kluwer Academic Publishers, 1997.

[SAS99] H. Schuldt, G. Alonso, and H.-J. Schek. Concurrency Control and Recovery for Transactional Processes. Technical report, Department of Computer Science, Swiss Federal Institute of Technology Zürich, 1999.

[SST98] H. Schuldt, H.-J. Schek, and M. Tresch. Coordination in CIM: Bringing Database Functionality to Application Systems. In *Proceedings of the* 5^{th} *European Concurrent Engineering Conference (ECEC'98)*, Erlangen, Germany, April 1998.

[SWY93] H.-J. Schek, G. Weikum, and H. Ye. Towards a Unifying Theory of Concurrency Control and Recovery. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'93)*, pages 300–311, June 1993.

[VHYBS98] R. Vingralek, H. Hasse-Ye, Y. Breitbart, and H.-J. Schek. Unifying concurrency control and recovery of transactions with semantically rich operations. *Theoretical Computer Science*, (190):363–396, 1998.

[WR92] H. Wächter and A. Reuter. *The ConTract Model*, chapter 7. In: [Elm92]. Morgan Kaufmann Publishers, 1992.

[ZNBB94] A. Zhang, M. Nodine, B. Bhargava, and O. Bukhres. Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems. In *Proceedings of the ACM SIGMOD Conference*, pages 67–78, 1994.

Appendix

A Proof of Theorem 1

Serializability: Assume that process schedule S is not serializable. Then, a conflict cycle has to exist of the form $P_i \ll_S P_j \ll_S \ldots \ll_S P_i$ in the committed projection of S. Therefore, this cycle also exists in the completed process schedule $\tilde{S}$. Thus, it follows that S cannot be reducible and therefore also not PRED.

Process-Recoverability: Assume that process schedule S is not process-recoverable. This can occur because one of the following four cases. In all these cases, the next non-compensatable activity of P_i succeeding a_{i_k} is denoted by a_{i_n} and a_{j_m} is the next non-compensatable activity of P_j succeeding a_{j_l}:

- **Case 1:** $a_{i_k} \ll_S a_{j_l} \ll_S a_{i_n} \ll_S a_{j_m} \ll_S C_j \ll_S C_i$. Consider the prefix S' of S that excludes C_i. The completion $\mathcal{C}(P_i)$ of P_i may contain an activity of the forward recovery path conflicting with any activity of process P_j. As these activities of $\mathcal{C}(P_i)$ are not known in advance, new conflicts are possible leading to S not being in PRED.
- **Case 2:** $a_{i_k} \ll_S a_{j_l} \ll_S a_{i_n} \ll_S a_{j_m} \ll_S C_j \ll_S A_i$. Consider the prefix S' of S that excludes A_i. This prefix is exactly the same as we considered in case 1. Thus, for the same reasons, a contradiction to the assumption of S being PRED arises.
- **Case 3:** $a_{i_k} \ll_S a_{j_l} \ll_S a_{i_n} \ll_S a_{j_m} \ll_S A_i \ll_S C_j$. Consider the completed process schedule $\tilde{S}$ of S. The completion $\mathcal{C}(P_i)$ of P_i may contain an activity of the forward recovery path conflicting with

any activity of process P_j. As these activities of $\mathcal{C}(P_i)$ are not known in advance, new conflicts with non-compensatable activities of P_j are possible leading to S not being in PRED.

Case 4: $a_{i_k} \ll_S a_{j_l} \ll_S a_{j_m} \ll_S a_{i_n}$. Consider the prefix S' of S that excludes a_{i_n}. Then, if a_{i_k} is compensatable, the compensation $a_{i_k}^{-1}$ of a_{i_k} has to be executed in the completed process schedule $\tilde{S}'$ of S'. This leads to a conflict cycle in $\tilde{S}'$ which cannot be eliminated as compensation of a_{j_l} is no longer available and contradicts with the initial assumption of S being PRED. If a_{i_k} is not compensatable, then activities of the completion $\mathcal{C}(P_i)$ of P_i may exist that introduce cyclic conflicts that cannot be eliminated. This also contradicts with the initial assumption. □

B Proof of Lemma 1

Assume that process schedule S is in PRED and that in S, a pair of conflicting activities a_{i_k} and a_{j_l} exists with $a_{i_k} \ll_S a_{j_l}$ and that process P_i is active.

1. Assume that a non-compensatable activity a_{j_m} is executed before P_i has terminated. Then, if some activity a_{i_n} of P_i has to be executed which is in conflict with a_{j_m}, they would have to be ordered in S as follows: $a_{j_m} \ll_S a_{i_n}$ leading to a conflict cycle in S. This cycle cannot be eliminated as:

   (i) a_{j_m} is a non-compensatable activity

   (ii) a_{i_k} cannot be compensated as this would, in turn, introduce another conflict cycle in the completed process schedule $\tilde{S}$ ($a_{i_k}^c \ll_{\tilde{S}} a_{j_l} \ll_{\tilde{S}} a_{j_m} \ll_{\tilde{S}} a_{i_k}^{-1}$)

   (iii) a_{j_l} cannot be compensated as it is followed by the non-compensating activity a_{j_m}.

   Therefore, process schedule S is not in RED and thus not in PRED leading to a contradiction with the initial assumption.

2. In this case, we have to differentiate whether a_{i_k} is compensatable or non-compensatable.

   (i) Assume that activity a_{i_k} is compensatable ($a_{i_k}^c$) while activity a_{j_l} is not compensatable ($a_{j_l}^p$ or $a_{j_l}^r$). Then, if the compensation of $a_{i_k}^c$ has to be considered in the completed process schedule $\tilde{S}$ (when process P_i is in $\mathcal{B} - \mathcal{REC}$), a conflict cycle by $a_{i_k}^c \ll_{\tilde{S}} a_{j_l} \ll_{\tilde{S}} a_{i_k}^{-1}$ appears. In this case, S is not in RED and also not in PRED leading to a contradiction with the initial assumption.

   (ii) Assume that both activity a_{i_k} and activity a_{j_l} are not compensatable (thus, both processes are in $\mathcal{F} - \mathcal{REC}$). As process P_i is active in S, further non-compensatable activities a_{i_n} may exist in the completion $\mathcal{C}(P_i)$ of P_i. Assume further that a_{i_n} is in conflict with a_{j_l}. Therefore, the order $a_{j_l} \ll_{\tilde{S}} a_{i_n}$ has to be imposed in the completed process schedule $\tilde{S}$ of S. This leads to cyclic conflicts in $\tilde{S}$ ($a_{i_k} \ll_{\tilde{S}} a_{j_l} \ll_{\tilde{S}} a_{i_n}$) that cannot be eliminated as all involved activities are non-compensatable. In this case, S is not in RED and also not in PRED which contradicts with the initial assumption. □

C Proof of Lemma 2

Assume that process schedule S is in PRED. Assume further that in the completed process schedule $\tilde{S}$ the compensating activities $a_{i_k}^{-1}$ and $a_{j_l}^{-1}$ are executed in the same order as the two conflicting activities $a_{i_k}^c$ and $a_{j_l}^c$. Then, in $\tilde{S}$, the following holds: $a_{i_k}^c \ll_{\tilde{S}} a_{j_l}^c \ll_{\tilde{S}} a_{i_k}^{-1} \ll_{\tilde{S}} a_{j_l}^{-1}$ leading to a conflict cycle that cannot be eliminated by one of the reduction rules. Therefore, S is not RED and thus also not PRED leading to a contradiction with the initial assumption. □

D Proof of Lemma 3

Suppose that process schedule S is in PRED with $a_{i_k}^c \in S$. Assume further that the two conflicting activities $a_{i_k}^{-1}$ and the non-compensatable activity $a_{j_l}^r$ are ordered in the completed process schedule $\tilde{S}$ as follows: $a_{j_l}^r \ll_{\tilde{S}} a_{i_k}^{-1}$. As commutativity is assumed to be perfect, a compensating activity has the same conflicts as its corresponding activity. Therefore, the conflict cycle $a_{i_k}^c \ll_{\tilde{S}} a_{j_l}^r \ll_{\tilde{S}} a_{i_k}^{-1}$ in $\tilde{S}$ exists and cannot be eliminated by the reduction rules and leads to the conclusion that S is not in RED and thus also not in PRED. This contradicts with the initial assumption. □