Appeared in: B. Eaglestone, B. C. Desai, J. Shao (eds.), Proc. of the 1998 Int. Database Engineering and Applications Symposium (IDEAS'98), July 8-10, 1998, Cardiff, Wales, UK, pages 34-43. IEEE Computer Science Press, Los Alamitos, CA, 1998.

Transitive Dependencies in Transaction Closures*

Kerstin Schwarz Can Türker Gunter Saake

Otto-von-Guericke-Universität Magdeburg
Institut für Technische und Betriebliche Informationssysteme
Postfach 4120, D–39016 Magdeburg, Germany
E-mail: {schwarz|tuerker|saake}@iti.cs.uni-magdeburg.de

Abstract

Complex applications consist of a large set of transactions which are interrelated. There are different kinds of dependencies among transactions of a complex application, e.g. termination or execution dependencies which are constraints on the occurrence of significant transaction events. In this paper, we analyze a set of (orthogonal) transaction dependencies. Here, we do not follow traditional approaches which consider advanced transaction structures as a certain kind of nested transactions. We introduce the notion of transaction closure as a generalization of nested transactions. A transaction closure comprises all transactions which are (transitively) initiated by one (root) transaction. By specifying dependencies among transactions of a transaction closure we are then able to define well-known transaction structures like nested transactions as well as advanced activity structures, e.g. workflows, in a common framework. In particular, we consider the transitivity property for all kinds of transaction dependencies discussed in this paper. Thus, we are able to conclude how two arbitrary transactions are transitively interrelated. This issue is fundamental for understanding the entire semantics of a complex application.

Keywords: *transaction closure, termination dependencies, object visibility constraints, transaction compensation, transitive dependencies.*

1 Introduction

Complex applications such as business processes or CSCW consist of sets of transactions which are interrelated. Transactions in these applications may be long-lived, may need to cooperate, or may require access to different autonomous databases. Furthermore, there are constraints on the execution order and the occurrence of termination events of related transactions, e.g. a certain transaction may only commit if another transaction fails. Thus, the complexity of such advanced applications is often so high that it is difficult to state how an application will behave if certain parts (transactions) fail. Here, a means is required for assisting the application designer to conclude which kinds of effects a certain transaction (transitively) has on other transactions.

The ACTA meta-model [3, 4] is a step in this direction by providing a framework for specifying advanced transaction models, e.g. nested transactions, Split transactions, and Sagas. For all these transaction models special dependencies are introduced (see [1] for a general discussion on specifying and enforcing intertransaction dependencies). ACTA allows to formally define dependencies among transactions. However, the current set of dependencies defined in ACTA requires some extensions in order to be usable as a generalized framework for describing and classifying arbitrary transaction models, particularly complex activity models.

Our goal is to find a (minimal) set of (orthogonal) fundamental transaction dependencies which are applicable according to real-world application semantics. For that, we investigate termination dependencies which are constraints on the occurrence of abort and commit events of related transactions and analyze how these dependencies can be combined. We show that there are some combinations which are not applicable. Thereafter, we investigate how these dependencies are influenced by different object visibility constraints and the concept of transaction compensation. In particular, we consider the transitivity property for all kinds of transaction dependencies discussed in this paper. Thus, we are able to derive the exact relationship between two arbitrary transactions. This issue is essential to detect *not explicitly specified* relationships among transactions.

Moreover, we introduce the notion of a *transaction closure* [13] as a generalized transaction structure consisting of a set of transactions which are (transitively) initiated by the same (root) transaction. Well-known advanced transactions such as nested transactions [12], flexible transactions [6], or ConTracts [15] can be seen as transaction closures with special dependencies among the transactions of this closure. A nested transaction, for example, is a transaction closure where the subtransactions must not leave the scope of its initiating transaction. In contrast to nested transactions, transaction closures can be used as a foundation for describing activity models [5] or workflow models [8].

*This research was partially supported by the German State Sachsen-Anhalt under FKZ 1987A/0025 and 1987/2527R.

The concepts of a transaction closure and of transaction dependencies with their transitivity properties are formally defined by using the ACTA framework. Thus, we are able to detect failures and contradictions in the specification of a transaction closure during the design process. Furthermore, the influence of transitive dependencies on a transaction closure can be simulated. This may help to detect superfluous parts (transactions) of a transaction closure definition, e.g. transactions which never can be committed due to transitive abort dependencies.

In summary, the concept of transaction closure together with the specified set of dependencies provide the basis for an assistance tool. Such a tool may help to understand the entire semantics of a complex application and thus it may support the design of better and more efficient applications.

The paper is organized as follows. In Section 2, we discuss the so-called termination dependencies which deal with valid combinations of termination events of related transactions. Execution dependencies are the topic of Section 3. In Section 4, the influence of object visibility constraints on termination dependencies is investigated. Thereafter, in Section 5, the effect of compensation aspects on termination dependencies is considered. The concept of transaction closure is introduced in Section 6. The exemplary application of transaction closures, especially the derivation of transitive dependencies in transaction closures, is shown in Section 7. Finally, the paper is concluded by an outlook on future work.

2 Termination Dependencies

Investigating constraints on the occurrence of the significant termination events commit and abort leads to different *termination dependencies*. In case of two transactions t_i and t_j, there are four possible combinations of termination events:

(1) both transactions abort (a_{t_i}, a_{t_j})

(2/3) one transaction commits whereas the other one aborts (a_{t_i}, c_{t_j})/(c_{t_i}, a_{t_j})

(4) both transactions commit (c_{t_i}, c_{t_j})

Obviously, constraints on the occurrence of these events lead to at most sixteen dependencies. Under the assumption that a transaction may be forced to abort but not to commit, the number of possibly reasonable termination dependencies is reduced to at most eight. This is due to the fact that the abortion of both transactions is considered as valid event combination for all dependencies. The termination event combinations of the eight dependencies are represented in Table 2. A ($\surd$) denotes that the corresponding termination event combination is valid. Otherwise, in case of (–) the event combination is not allowed to occur.

We identify five of the eight dependencies as applicable according to real-world application semantics. The remaining three combinations of termination events are inapplicable according to real-world application semantics. The combination where only the abortion of both transaction is valid (see the first dependency in Table 2) makes no sense because in this case no transaction is allowed to commit. In other words, if transactions are only allowed to abort, they do not need to be executed. The next two combinations are also not reasonable because in these cases always one of the related two transactions is not allowed to commit, independently of the termination event of the other transaction (see the second and third dependencies in Table 2).

t_i	t_j	1	2	3	4	5	6	7	8
a_{t_i}	a_{t_j}	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
a_{t_i}	c_{t_j}	—	$\surd$	—	—	—	$\surd$	$\surd$	$\surd$
c_{t_i}	a_{t_j}	—	—	$\surd$	—	$\surd$	—	$\surd$	$\surd$
c_{t_i}	c_{t_j}	—	—	—	$\surd$	$\surd$	$\surd$	—	$\surd$

Table 2. Termination Event Combinations

After having discussed the non-reasonable termination event combinations, we will now formally define the termination dependencies which are based on reasonable termination events combinations. For our definitions we use the ACTA formalism [3, 4], including the fundamental dependency definitions such as the *abort dependency* and *exclusive dependency* and the notion of a history of transactions (in the following formulas denoted as H).

Abort Dependency $(t_i\,\mathcal{AD}\,t_j)$**.** If transaction t_j aborts, then transaction t_i has to abort, too:

$$abort_{t_j} \in H \Rightarrow abort_{t_i} \in H$$

Exclusive Dependency $(t_i\,\mathcal{ED}\,t_j)$**.** If transaction t_j commits, then transaction t_i has to abort:

$$commit_{t_j} \in H \Rightarrow (begin_{t_i} \in H \Rightarrow abort_{t_i} \in H)$$

As an extension of the dependencies specified in ACTA, we define the transitive closure property for all dependencies. In doing so, we are able to analyze the influences of dependencies among transactions which are only indirectly interrelated by further transactions.

A dependency between two transactions t_i and t_j which requires that both transactions either abort or commit together is called *vital-dependent*.

Definition 1 (Vital-Dependent) *Two different transactions t_i and t_j are* vital-dependent *for each other iff both transactions are (transitively)*[1] abort dependent *on each other:*

$$\begin{aligned} vital\_dep(t_i, t_j) \quad :\Leftrightarrow \quad & ((t_i\,\mathcal{AD}\,t_j) \wedge (t_j\,\mathcal{AD}\,t_i)) \vee \\ & (\exists t_k : vital\_dep(t_i, t_k) \wedge vital\_dep(t_k, t_j)) \end{aligned}$$

The vital-dependent dependency is (as the name suggests) a combination of the dependencies *vital* and *dependent*.

[1] Because of space restrictions we cannot attach the proof of the transitivity property in the appendix. Therefore, please refer to [14].

t_i	t_j	$vital\_dep(t_i,t_j)$	$vital(t_i,t_j)$	$dep(t_i,t_j)$	$exc(t_i,t_j)$	$indep(t_i,t_j)$
a_{t_i}	a_{t_j}	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
a_{t_i}	c_{t_j}	—	—	$\surd$	$\surd$	$\surd$
c_{t_i}	a_{t_j}	—	$\surd$	—	$\surd$	$\surd$
c_{t_i}	c_{t_j}	$\surd$	$\surd$	$\surd$	—	$\surd$

Table 1. Reasonable Termination Dependencies between two Transactions t_i and t_j

Definition 2 (Vital) *A transaction t_i is* vital *for another transaction t_j iff t_j is (transitively)* abort dependent *on t_i:*

$$\begin{aligned} vital(t_i,t_j) \quad :\Leftrightarrow \quad & (t_j\,\mathcal{AD}\,t_i) \vee \\ & (\exists t_k : (vital\_dep(t_i,t_k) \wedge vital(t_k,t_j)) \vee \\ & \quad (vital(t_i,t_k) \wedge vital\_dep(t_k,t_j)) \vee \\ & \quad (vital(t_i,t_k) \wedge vital(t_k,t_j))) \end{aligned}$$

Thus, the *vital* dependency between two transactions t_i and t_j concerns the case where the abortion of transaction t_i leads to the abortion of transaction t_j. In contrast to the vital dependency, a *dependent* transaction t_i has to abort if transaction t_j aborts. Thus, the termination event of one transaction is necessary for an acceptable outcome according to the semantics of the application. This is the vital and dependent, respectively, transaction t_i. In contrast, the results of transaction t_j are not essential for the application.

Definition 3 (Dependent) *A transaction t_i is* dependent *on another transaction t_j iff t_i is (transitively)* abort dependent *on t_j:*

$$\begin{aligned} dep(t_i,t_j) \quad :\Leftrightarrow \quad & (t_i\,\mathcal{AD}\,t_j) \vee \\ & (\exists t_k : (vital\_dep(t_i,t_k) \wedge dep(t_k,t_j)) \vee \\ & \quad (dep(t_i,t_k) \wedge vital\_dep(t_k,t_j)) \vee \\ & \quad (dep(t_i,t_k) \wedge dep(t_k,t_j))) \end{aligned}$$

A completely different dependency is the *exclusive* dependency. Here, only one of the transactions is allowed to finish successfully. This dependency may be useful in real-time applications where two alternative transactions are executed in parallel and the results of the one which finishes first are accepted and the other one is aborted after that.

Definition 4 (Exclusive) *Two different transactions t_i and t_j are* exclusive *for each other iff both transactions are (transitively)* exclusive dependent *on each other:*

$$\begin{aligned} exc(t_i,t_j) \quad :\Leftrightarrow \quad & ((t_i\,\mathcal{ED}\,t_j) \wedge (t_j\,\mathcal{ED}\,t_i)) \vee \\ & (\exists t_k : (vital\_dep(t_i,t_k) \wedge exc(t_k,t_j)) \vee \\ & \quad (exc(t_i,t_k) \wedge vital\_dep(t_k,t_j)) \vee \\ & \quad (dep(t_i,t_k) \wedge exc(t_k,t_j)) \vee \\ & \quad (exc(t_i,t_k) \wedge vital(t_k,t_j))) \end{aligned}$$

Our fifth dependency concerns the case where each combination of transaction termination events is valid. Therefore, the involved transactions are denoted as *independent*. Concurrently executed transactions without any constraints on the execution order and termination events are independent.

Definition 5 (Independent) *Two different transactions t_i and t_j are* independent *if all termination event combinations of t_i and t_j are allowed. This case is denoted as $indep(t_i,t_j)$.*

The termination event combinations of the dependencies defined so far are summarized in Table 1.

After having a closer look on the definitions above, we can conclude that a vital dependency between transactions t_i and t_j ($vital(t_i,t_j)$) is equivalent to the $dep(t_j,t_i)$ dependency between transaction t_j and t_i and vice versa. In other words, transaction t_i is vital for t_j if and only if t_j is dependent on t_i. In contrast, the other dependencies are symmetrical (by definition).

Theorem 6 *The following relationships hold for termination dependencies between two transactions t_i and t_j:*

$$\begin{aligned} vital\_dep(t_i,t_j) &\equiv vital\_dep(t_j,t_i) \\ vital(t_i,t_j) &\equiv dep(t_j,t_i) \\ exc(t_i,t_j) &\equiv exc(t_j,t_i) \\ indep(t_i,t_j) &\equiv indep(t_j,t_i) \end{aligned}$$

A conclusion which directly follows from Theorem 6 is that either *vital* or *dependent* is not required because these dependencies can be substituted by each other. This observation is true if we do not consider combinations of termination dependencies with other kinds of dependencies. As we will see in Section 4, the combination of termination dependencies with object visibility constraints leads to different results for $vital(t_i,t_j)$ and $dep(t_i,t_j)$. Therefore, we explicitly distinguish these two termination dependencies.

The following example shall clarify the derivation of transitive dependencies.

Example 7 *Let t_1, t_2, t_3, and t_4 be transactions which are connected by the following dependencies:*

$$vital\_dep(t_1,t_2) \wedge dep(t_2,t_3) \wedge vital(t_1,t_4)$$

This scenario is depicted in Figure 1 where the arrows denote the direction of the abort dependencies. For example, $t_1 \longrightarrow t_4$ means that the abortion of transaction t_1 leads to the abortion of t_4. Hence, the termination dependencies (used in this example) are illustrated as follows:

$$\begin{aligned} vital(t_i,t_j) &\quad \textit{corresponds to} \quad t_i \longrightarrow t_j \\ dep(t_i,t_j) &\quad \textit{corresponds to} \quad t_i \longleftarrow t_j \\ vital\_dep(t_i,t_j) &\quad \textit{corresponds to} \quad t_i \longleftrightarrow t_j \end{aligned}$$

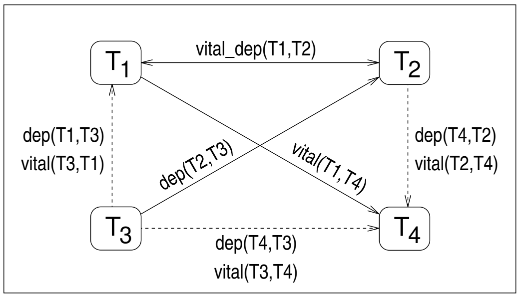

Figure 1. Derived Termination Dependencies

From our dependency definitions and Theorem 6 we can now derive the following transitive dependencies among the transactions t_1, t_2, t_3 and t_4:

$$\begin{aligned} & vital\_dep(t_1, t_2) \wedge vital(t_1, t_4) \\ & \equiv vital\_dep(t_2, t_1) \wedge vital(t_1, t_4) \implies vital(t_2, t_4) \\ & \qquad \equiv dep(t_4, t_2) \\ & vital\_dep(t_1, t_2) \wedge dep(t_2, t_3) \implies dep(t_1, t_3) \\ & \qquad \equiv vital(t_3, t_1) \\ & vital(t_3, t_1) \wedge vital(t_1, t_4) \implies vital(t_3, t_4) \\ & \qquad \equiv dep(t_4, t_3) \end{aligned}$$

In Figure 1, these transitive dependencies are represented by the dashed arrows.

3 Execution Dependencies

In general, there are also constraints on the significant events begin, commit and abort, respectively, of a set of transactions. We distinguish the following kinds of *execution dependencies*: *sequential* (as generalization of *sequential after abort* and *sequential after commit*) and *parallel*.

For specifying these dependencies we require the following fundamental dependencies defined in ACTA:

Serial Dependency $(t_i \, \mathcal{SD} \, t_j)$**.** A transaction t_i cannot begin executing until transaction t_j either commits or aborts:

$$(begin_{t_i} \in H) \Rightarrow ((commit_{t_j} \to begin_{t_i}) \vee (abort_{t_j} \to begin_{t_i}))$$

Begin-on-Abort Dependency $(t_i \, \mathcal{BAD} \, t_j)$**.** A transaction t_i cannot begin its execution until transaction t_j aborts:

$$(begin_{t_i} \in H) \Rightarrow (abort_{t_j} \to begin_{t_i})$$

Begin-on-Commit Dependency $(t_i \, \mathcal{BCD} \, t_j)$**.** This dependency denotes that a transaction t_i cannot begin executing until transaction t_j commits:

$$(begin_{t_i} \in H) \Rightarrow (commit_{t_j} \to begin_{t_i})$$

Using this fundamental dependencies we are able to define several execution dependencies. First, we specify the general *sequential* dependency.

Definition 8 (Sequential) *A transaction t_i is* sequential dependent *on another transaction t_j iff t_j is (transitively)* serial dependent *on t_i:*

$$\begin{aligned} seq(t_i, t_j) \quad :\Leftrightarrow \quad & (t_j \, \mathcal{SD} \, t_i) \vee \\ & (\exists t_k : seq(t_i, t_k) \wedge seq(t_k, t_j)) \end{aligned}$$

The sequential dependency can be further refined. A transaction can be allowed to start only after the abortion or after the commit, respectively, of another transaction.

Definition 9 (Sequential-Abort) *A transaction t_i is* sequential-abort dependent *on another transaction t_j iff t_j is (transitively)* begin-on-abort dependent *on t_i:*

$$\begin{aligned} seq\_a(t_i, t_j) \quad :\Leftrightarrow \quad & (t_j \, \mathcal{BAD} \, t_i) \vee \\ & (\exists t_k : seq\_a(t_i, t_k) \wedge seq(t_k, t_j)) \end{aligned}$$

Definition 10 (Sequential-Commit) *A transaction t_i is* sequential-commit dependent *on another transaction t_j iff t_j is (transitively)* begin-on-commit *dependent on t_i:*

$$\begin{aligned} seq\_c(t_i, t_j) \quad :\Leftrightarrow \quad & (t_j \, \mathcal{BCD} \, t_i) \vee \\ & (\exists t_k : seq\_c(t_i, t_k) \wedge seq(t_k, t_j)) \end{aligned}$$

In contrast to sequential executed transactions, a *parallel* transaction starts before the termination of another one. We define a *begin-before-terminate dependency* to specify the parallel dependency.

Begin-before-Terminate Dependency $(t_i \, \mathcal{BBT} \, t_j)$**.** A transaction t_i cannot terminate (commit or abort) until transaction t_j begin executing:

$$(\epsilon \in H) \Rightarrow (begin_{t_j} \to \epsilon), where \; \epsilon \in \{commit_{t_i}, abort_{t_i}\}$$

Definition 11 (Parallel) *A transaction t_i is* parallel *for another transaction t_j if and only if both transactions are begin-before-terminate dependent on each other:*

$$par(t_i, t_j) \quad :\Leftrightarrow \quad (t_i \, \mathcal{BBT} \, t_j) \wedge (t_j \, \mathcal{BBT} \, t_i)$$

4 Object Visibility Constraints

In this section, we investigate the effects of object visibility constraints on termination dependencies. Object visibility is of major concern in the context of isolated and atomic transaction executions. In general, there are two ways how to deal with the results of a transaction:

1. The results of a transaction are made visible to only one transaction, e.g. in a closed nested transaction model the effects of a subtransaction are made visible to only the parent transaction (in case of a commit of the subtransaction).

t_i	t_j	$vital\_dep_d(t_i,t_j)$	$vital_d(t_i,t_j)$	$dep_d(t_i,t_j)$	$exc_d(t_i,t_j)$	$indep_d(t_i,t_j)$
a_{t_i}	a_{t_j}	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
a_{t_i}	c_{t_j}	—	—	$a_{t_i} \to c_{t_j}$	$a_{t_i} \to c_{t_j}$	$a_{t_i} \to c_{t_j}$
c_{t_i}	a_{t_j}	—	$\surd$	—	$\surd$	$\surd$
c_{t_i}	c_{t_j}	$c_{t_i} \to c_{t_j}$	$c_{t_i} \to c_{t_j}$	$c_{t_i} \to c_{t_j}$	—	$c_{t_i} \to c_{t_j}$

Table 3. Combining Termination Dependencies with the Delegating Property

2. All effects of a transaction are made visible to all other transactions with the commit of the transaction.

In the first case, a transaction delegates the responsibility for committing or aborting its effects to another transaction. This delegation operation is defined as follows:

Definition 12 (Delegate) *A* delegating *transaction t_j delegates the responsibility for committing or aborting its access set to another transaction t_i (called the* receiving *transaction) by using the operation $delegate_{t_j}[t_i]$. After $delegate_{t_j}[t_i]$ the access set of t_i includes the access set of t_j.*

A delegating transaction can only delegate its access set to an active (non-terminated) transaction. In other words, the termination of the delegating transaction must precede the termination of the receiving transaction. So, the receiving transaction has to wait for the termination of the delegating transaction to commit. To enforce this order of termination events, we define the following *commit-on-termination dependency*.

Commit-on-Termination Dependency ($t_i\, \mathcal{CTD}\, t_j$).
A transaction t_i cannot commit until transaction t_j either commits or aborts.

$$commit_{t_i} \in H \quad \Rightarrow \quad ((commit_{t_j} \to commit_{t_i}) \vee (abort_{t_j} \to commit_{t_i}))$$

The definitions of the delegate operation and of the commit-on-termination dependency enable us to define the delegating dependency.

Definition 13 (Delegating) *A transaction t_i is a* delegating *transaction from the viewpoint of another transaction t_j iff t_j is (transitively)* commit-on-termination dependent *on t_i and t_i (transitively) delegates its access set to t_j:*

$$\begin{aligned} del(t_i,t_j) \quad :\Leftrightarrow \quad & ((t_j\, \mathcal{CTD}\, t_i) \wedge \\ & (commit_{t_i} \in H \Leftrightarrow delegate_{t_i}[t_j] \in H)) \vee \\ & (\exists t_k : del(t_i,t_k) \wedge del(t_k,t_j)) \end{aligned}$$

A transaction which makes all its effects visible to all other transactions with its commit is called a *releasing* transaction. In other words, a releasing transaction does not delegates its access set to another transaction; the access set is released for all other transactions.

Definition 14 (Releasing) *A transaction t_j is a releasing transaction from the viewpoint of transaction t_i if the access set of t_j is visible with the commit of t_j.*

Definition 15 (Combined Dependencies) *Combining the termination dependencies introduced in Section 2 with the delegating dependency leads to the following dependencies:*

$$\begin{aligned} vital\_dep_d(t_i,t_j) \quad &:\Leftrightarrow \quad del(t_i,t_j) \wedge vital\_dep(t_i,t_j) \\ vital_d(t_i,t_j) \quad &:\Leftrightarrow \quad del(t_i,t_j) \wedge vital(t_i,t_j) \\ dep_d(t_i,t_j) \quad &:\Leftrightarrow \quad del(t_i,t_j) \wedge dep(t_i,t_j) \\ exc_d(t_i,t_j) \quad &:\Leftrightarrow \quad del(t_i,t_j) \wedge exc(t_i,t_j) \\ indep_d(t_i,t_j) \quad &:\Leftrightarrow \quad del(t_i,t_j) \wedge indep(t_i,t_j) \end{aligned}$$

Table 3 summarizes the effects of the combined dependencies on the possible combinations of termination events. Comparing Table 1 and 3, we can see that the delegating property influences the second and the last row of Table 1, i.e., all valid combinations of termination events where the receiving transaction t_j commits. These termination event combinations are restricted by the constraint that the commit of the delegating transaction must precede the commit or abort, respectively, of the receiving transaction. The reason for this restriction is that the delegating property bases on the commit-on-termination dependency which determines a given termination order.

In Table 3 we further see that there is a difference between the dependencies $vital_d(t_i,t_j)$ and $dep_d(t_j,t_i)$. Because of the commit-on-termination dependency, these dependencies cannot be used interchangeable. In contrast to the basic termination dependencies, the termination order of the transactions is fixed by the delegating dependency:

Theorem 16 *For two transactions t_i and t_j the following dependency relationships hold:*

$$\begin{aligned} vital\_dep_d(t_i,t_j) \quad &\not\equiv \quad vital\_dep_d(t_j,t_i) \\ vital_d(t_i,t_j) \quad &\not\equiv \quad dep_d(t_j,t_i) \\ exc_d(t_i,t_j) \quad &\not\equiv \quad exc_d(t_j,t_i) \\ indep_d(t_i,t_j) \quad &\not\equiv \quad indep_d(t_j,t_i) \end{aligned}$$

5 Influence of Transaction Compensation on Termination Dependencies

In the previous section, we saw that the delegating dependency put further constraints on the termination dependencies. In this section, we will see that the termination dependencies can be weakened when transaction compensation is supported. Transaction compensation is the ability to semantically undo the effects of a transaction t_i by a *compensating* transaction $comp_{t_i}$ to achieve semantic atomicity. If there exists such a compensating transaction $comp_{t_i}$ for a transaction t_i, t_i is denoted as *compensatable*. Obviously, the aspect of compensation makes only sense for releasing transactions.

The basic termination dependencies defined in Section 2 base on the abort dependency and the exclusive dependency, respectively. If transaction compensation is considered, these dependencies are weakened to the *weak-abort dependency*[2] and the *weak-exclusive dependency*, respectively.

Weak-Abort Dependency $(t_i\, \mathcal{WAD}\, t_j)$**.** If transaction t_i aborts and transaction t_j commits, then the commit of t_i precedes the abort of t_j and the compensating transaction $comp_{t_j}$ of t_j has to commit, too.

$$(abort_{t_j} \in H) \Rightarrow ((commit_{t_i} \in H) \Rightarrow (commit_{t_i} \rightarrow abort_{t_j} \wedge (commit_{comp_{t_i}} \in H)))$$

In contrast to an abort dependent transaction, a weak-abort dependent transaction can commit without waiting for the commit of the other transaction. In case the other transaction aborts, the "committed" effects of the weak-abort dependent transaction are semantically undone by executing a compensating transaction.

Weak-Exclusive Dependency $(t_i\, \mathcal{WED}\, t_j)$**.** If both transactions t_i and t_j commit, then the compensating transaction of t_i ($comp_{t_i}$) or of t_j ($comp_{t_j}$) has to commit, too.

$$(commit_{t_j} \in H) \Rightarrow ((commit_{t_i} \in H) \Rightarrow ((commit_{comp_{t_i}} \in H) \vee (commit_{comp_{t_j}} \in H)))$$

In comparison to the exclusive dependency, the weak-exclusive dependency only demands that (at least) one of the compensating transactions of the related transactions t_i and t_j must be committed in case both t_i and t_j commit. Thus, the case where both transactions can commit is allowed under the restriction that one of these transactions is compensatable.

By using the weakened versions of the abort dependency and the exclusive dependency, we now redefine the termination dependencies under the consideration that the related transactions are compensatable (at least one of them).

For the symmetrical dependency $vital\_dep(t_i, t_j)$ between two transactions t_i and t_j we have to distinguish three cases:

- If only t_i is compensatable, the corresponding dependency is denoted as $vital\_dependent_{\blacktriangleleft}$.
- If only t_j is compensatable, the corresponding dependency is denoted as $vital\_dependent_{\blacktriangleright}$.
- If both transactions t_i and t_j are compensatable, the corresponding dependency is denoted as $vital\_dependent_{\blacklozenge}$.

[2]Please note that our definition of the weak-abort dependency differs from the weak-abort dependency $(t_i\, \mathcal{WD}\, t_j)$ defined in [4].

Definition 17 (Vital-Dependent$_{\blacklozenge}$) *Two different transactions t_i and t_j are* vital-dependent$_{\blacklozenge}$ *on each other iff both transactions are (transitively)* weak-abort dependent *on each other:*

$$\begin{aligned} vital\_dep_{\blacklozenge}(t_i, t_j) \quad :\Leftrightarrow \quad & ((t_i\, \mathcal{WAD}\, t_j) \wedge (t_j\, \mathcal{WAD}\, t_i)) \vee \\ & (\exists t_k : (vital\_dep_{\blacklozenge}(t_i, t_k) \wedge vital\_dep_{\blacklozenge}(t_k, t_j))) \end{aligned}$$

Definition 18 (Vital-Dependent$_{\blacktriangleleft}$) *Two different transactions t_i and t_j are* vital-dependent$_{\blacktriangleleft}$ *on each other iff t_i is (transitively)* weak-abort dependent *on t_j and t_j is (transitively)* abort dependent *on t_i:*

$$\begin{aligned} vital\_dep_{\blacktriangleleft}(t_i, t_j) \quad :\Leftrightarrow \quad & ((t_i\, \mathcal{WAD}\, t_j) \wedge (t_j\, \mathcal{AD}\, t_i)) \vee \\ & (\exists t_k : (vital\_dep_{\blacktriangleleft}(t_i, t_k) \wedge vital\_dep(t_k, t_j)) \vee \\ & (vital\_dep_{\blacklozenge}(t_i, t_k) \wedge vital\_dep_{\blacktriangleleft}(t_k, t_j))) \end{aligned}$$

Definition 19 (Vital-Dependent$_{\blacktriangleright}$) *Two different transactions t_i and t_j are* vital-dependent$_{\blacktriangleright}$ *on each other iff t_j is (transitively)* weak-abort dependent *on t_i and t_i is (transitively)* abort dependent *on t_j:*

$$\begin{aligned} vital\_dep_{\blacktriangleright}(t_i, t_j) \quad :\Leftrightarrow \quad & (t_i\, \mathcal{AD}\, t_j) \wedge (t_j\, \mathcal{WAD}\, t_i) \vee \\ & (\exists t_k : (vital\_dep(t_i, t_k) \wedge vital\_dep_{\blacktriangleright}(t_k, t_j)) \vee \\ & (vital\_dep_{\blacktriangleright}(t_i, t_k) \wedge vital\_dep_{\blacklozenge}(t_k, t_j))) \end{aligned}$$

Definition 20 (Vital$_{\blacktriangleright}$) *A transaction t_i is* vital$_{\blacktriangleright}$ *for another transaction t_j iff t_j is (transitively)* weak-abort dependent *on t_i:*

$$\begin{aligned} vital_{\blacktriangleright}(t_i, t_j) \quad :\Leftrightarrow \quad & (t_j\, \mathcal{WAD}\, t_i) \vee \\ & (\exists t_k : (vital(t_i, t_k) \wedge vital\_dep_{\blacktriangleright}(t_k, t_j)) \vee \\ & (vital\_dep_{\blacktriangleleft}(t_i, t_k) \wedge vital_{\blacktriangleright}(t_k, t_j)) \vee \\ & (vital_{\blacktriangleright}(t_i, t_k) \wedge vital\_dep_{\blacklozenge}(t_k, t_j)) \vee \\ & (vital\_dep(t_i, t_k) \wedge vital_{\blacktriangleright}(t_k, t_j)) \vee \\ & (vital(t_i, t_k) \wedge vital_{\blacktriangleright}(t_k, t_j)) \vee \\ & (vital_{\blacktriangleright}(t_i, t_k) \wedge vital_{\blacktriangleright}(t_k, t_j)) \vee \\ & (vital\_dep_{\blacktriangleright}(t_i, t_k) \wedge vital_{\blacktriangleright}(t_k, t_j)) \vee \\ & (vital\_dep_{\blacklozenge}(t_i, t_k) \wedge vital_{\blacktriangleright}(t_k, t_j))) \end{aligned}$$

Definition 21 (Dependent$_{\blacktriangleleft}$) *A transaction t_j is* dependent$_{\blacktriangleleft}$ *on another transaction t_i iff t_i is (transitively)* weak-abort dependent *on t_j:*

$$\begin{aligned} dep_{\blacktriangleleft}(t_i, t_j) \quad :\Leftrightarrow \quad & (t_i\, \mathcal{WAD}\, t_j) \vee \\ & (\exists t_k : (vital\_dep_{\blacktriangleleft}(t_i, t_k) \wedge dep(t_k, t_j)) \vee \\ & (dep_{\blacktriangleleft}(t_i, t_k) \wedge vital\_dep_{\blacktriangleright}(t_k, t_j)) \vee \\ & (vital\_dep_{\blacklozenge}(t_i, t_k) \wedge dep_{\blacktriangleleft}(t_k, t_j)) \vee \\ & (dep_{\blacktriangleleft}(t_i, t_k) \wedge vital\_dep(t_k, t_j)) \vee \\ & (dep_{\blacktriangleleft}(t_i, t_k) \wedge dep(t_k, t_j)) \vee \\ & (dep_{\blacktriangleleft}(t_i, t_k) \wedge dep_{\blacktriangleleft}(t_k, t_j)) \vee \\ & (dep_{\blacktriangleleft}(t_i, t_k) \wedge vital\_dep_{\blacktriangleleft}(t_k, t_j)) \vee \\ & (dep_{\blacktriangleleft}(t_i, t_k) \wedge vital\_dep_{\blacklozenge}(t_k, t_j))) \end{aligned}$$

t_i	t_j	$vital\_dep_{\blacklozenge}(t_i,t_j)$	$vital\_dep_{\blacktriangleleft}(t_i,t_j)$	$vital\_dep_{\blacktriangleright}(t_i,t_j)$
a_{t_i}	a_{t_j}	$\surd$	$\surd$	$\surd$
a_{t_i}	c_{t_j}	$c_{t_j} \rightarrow a_{t_i} \wedge c_{comp_{t_j}}$	—	$c_{t_j} \rightarrow a_{t_i} \wedge c_{comp_{t_j}}$
c_{t_i}	a_{t_j}	$c_{t_i} \rightarrow a_{t_j} \wedge c_{comp_{t_i}}$	$c_{t_i} \rightarrow a_{t_j} \wedge c_{comp_{t_i}}$	—
c_{t_i}	c_{t_j}	$\surd$	$\surd$	$\surd$

t_i	t_j	$vital_{\blacktriangleright}(t_i,t_j)$	$dep_{\blacktriangleleft}(t_i,t_j)$	$exc_{\blacklozenge}(t_i,t_j)$
a_{t_i}	a_{t_j}	$\surd$	$\surd$	$\surd$
a_{t_i}	c_{t_j}	$c_{t_j} \rightarrow a_{t_i} \wedge c_{comp_{t_j}}$	$\surd$	$\surd$
c_{t_i}	a_{t_j}	$\surd$	$c_{t_i} \rightarrow a_{t_j} \wedge c_{comp_{t_i}}$	$\surd$
c_{t_i}	c_{t_j}	$\surd$	$\surd$	$c_{comp_{t_i}} \vee c_{comp_{t_j}}$

Table 4. Relaxed Termination Dependencies

Definition 22 (Exclusive$_{\blacklozenge}$) *A transaction t_j is* exclusive$_{\blacklozenge}$ *for another transaction t_i iff t_i is (transitively)* weak-exclusive dependent *on t_j:*

$$\begin{aligned} exc_{\blacklozenge}(t_i,t_j) \quad :\Leftrightarrow \quad & (t_i\,\mathcal{WED}\,t_j) \vee \\ & (\exists t_k : (vital\_dep_{\blacktriangleleft}(t_i,t_k) \wedge exc(t_k,t_j)) \vee \\ & \quad (exc(t_i,t_k) \wedge vital\_dep_{\blacktriangleright}(t_k,t_j)) \vee \\ & \quad (exc(t_i,t_k) \wedge vital_{\blacktriangleright}(t_k,t_j)) \vee \\ & \quad (dep_{\blacktriangleleft}(t_i,t_k) \wedge exc(t_k,t_j)) \vee \\ & \quad (vital\_dep_{\blacklozenge}(t_i,t_k) \wedge exc_{\blacklozenge}(t_k,t_j)) \vee \\ & \quad (exc_{\blacklozenge}(t_i,t_k) \wedge vital\_dep_{\blacklozenge}(t_k,t_j)) \vee \\ & \quad (vital\_dep_{\blacktriangleright}(t_i,t_k) \wedge exc_{\blacklozenge}(t_k,t_j)) \vee \\ & \quad (exc_{\blacklozenge}(t_i,t_k) \wedge vital\_dep_{\blacktriangleleft}(t_k,t_j)) \vee \\ & \quad (vital\_dep(t_i,t_k) \wedge exc_{\blacklozenge}(t_k,t_j)) \vee \\ & \quad (exc_{\blacklozenge}(t_i,t_k) \wedge vital\_dep(t_k,t_j)) \vee \\ & \quad (vital\_dep_{\blacktriangleleft}(t_i,t_k) \wedge exc_{\blacklozenge}(t_k,t_j)) \vee \\ & \quad (exc_{\blacklozenge}(t_i,t_k) \wedge vital_{\blacktriangleright}(t_k,t_j)) \vee \\ & \quad (exc_{\blacklozenge}(t_i,t_k) \wedge vital\_dep_{\blacktriangleright}(t_k,t_j)) \vee \\ & \quad (dep(t_i,t_k) \wedge exc_{\blacklozenge}(t_k,t_j)) \vee \\ & \quad (dep_{\blacktriangleleft}(t_i,t_k) \wedge exc_{\blacklozenge}(t_k,t_j)) \vee \\ & \quad (exc_{\blacklozenge}(t_i,t_k) \wedge vital(t_k,t_j))) \end{aligned}$$

Table 4 gives an overview over the termination dependencies in combination with the compensation aspect. Comparing Table 4 with Table 1, all disallowed fields (–) of the basic termination dependencies are replaced by a constraint on the termination order ($\rightarrow$) of the corresponding transactions (except of two fields in $vital\_dep_{\blacktriangleleft}(t_i,t_j)$ and $vital\_dep_{\blacktriangleright}(t_i,t_j)$, respectively). This is caused by the weak-abort and weak-exclusive dependencies.

From the Definitions 17 to 22, we can derive the following theorem.

Theorem 23 *The following relationships hold for dependencies between two transactions t_i and t_j:*

$$\begin{aligned} vital\_dep_{\blacklozenge}(t_i,t_j) &\equiv vital\_dep_{\blacklozenge}(t_j,t_i) \\ vital\_dep_{\blacktriangleright}(t_i,t_j) &\equiv vital\_dep_{\blacktriangleleft}(t_j,t_i) \\ \exists comp_{t_i} : \nexists comp_{t_j} : vital_{\blacktriangleright}(t_i,t_j) &\equiv dep(t_j,t_i) \\ \nexists comp_{t_i} : \exists comp_{t_j} : vital(t_i,t_j) &\equiv dep_{\blacktriangleleft}(t_j,t_i) \\ \exists comp_{t_i} : \exists comp_{t_j} : vital_{\blacktriangleright}(t_i,t_j) &\equiv dep_{\blacktriangleleft}(t_j,t_i) \\ exc_{\blacklozenge}(t_i,t_j) &\equiv exc_{\blacklozenge}(t_j,t_i) \end{aligned}$$

6 Transaction Closures

Traditionally, advanced transaction models base on the idea of nesting transactions. A nested transaction [12] is a transaction consisting of smaller transactions called *subtransactions*. These subtransactions are initiated within a transaction. The initiating transaction is called *parent*. A subtransaction itself may consist of smaller subtransactions, i.e., a nested transaction forms a transaction tree. However, activity models [5] or workflow models [11] require a more general framework for describing a set of related transactions. The same holds for execution models of active database systems [10].

Therefore, we have introduced the notion of a *transaction closure* [13] as a generalized transaction structure. A transaction closure consists of a set of transactions which are transitively initiated by the same (root) transaction, i.e., there are parent-child relationships between transaction pairs of a transaction closure. The transactions of a transaction closure are further interrelated by the dependencies discussed in the previous sections. For example, a subtransaction in a transaction closure may leave the scope of its parent transaction. By using transaction closures (in connection with the various dependency types) we have a uniform framework for describing traditional as well as advanced transaction models, particularly activity and workflow models. A closed nested transaction [12], for example, is a transaction closure where the subtransactions are vital-dependent or vital for their parents including the delegating property ($vital\_dep_d(t_i,t_j)$ or $vital_d(t_i,t_j)$).

For the formal definition of the notion of a transaction closure we require some basic definitions.

Definition 24 *The following self-explanatory functions and* *predicates describe general relationships between a transaction and its initiator:*

$$\begin{array}{rcl} parent(t_i, t_j) & := & (\, t_i \text{ is parent of } t_j \,) \\ root(t_j) & := & (\, t_j \text{ has no parent} \,) \\ ancestor(t_i, t_j) & := & (\, parent(t_i, t_j) \vee \\ & & (\exists t_k : ancestor(t_i, t_k) \wedge parent(t_k, t_j)\,)\,) \end{array}$$

The relationship between the initiation of two transactions can be expressed by a *begin dependency*.

Begin Dependency $(t_i \, \mathcal{BD} \, t_j)$**.** A transaction t_i can be initiated only if transaction t_j is already initiated.

$$begin_{t_i} \in H \Rightarrow (begin_{t_j} \rightarrow begin_{t_i})$$

Definition 25 (Transaction Closures) *Suppose tc_m and tc_n denote two (different) transaction closures and let t_i and t_j be two transactions:*

1. *Transactions of two different transaction closures are always independent:*

   $$\forall t_i \in tc_m, \forall t_j \in tc_n, tc_m \neq tc_n : indep(t_i, t_j)$$

2. *Each transaction closure has* exactly one[3] *root transaction:* $\forall tc_m : \exists!\, t_i \in tc_m : root(t_i)$

3. *Each non-root transaction has* exactly one *parent transaction:*

   $$\begin{array}{l} \forall tc_m : \forall t_j \in tc_m : \neg root(t_j) \Rightarrow \\ \qquad (\exists!\, t_i \in tc_m : parent(t_i, t_j)) \end{array}$$

4. *Each transaction closure is acyclic:*

   $$\forall tc_m : \nexists t_i \in tc_m : ancestor(t_i, t_i)$$

5. *The initiation of a transaction must follow the initiation of the parent:*

   $$\begin{array}{l} \forall tc_m : \forall t_j \in tc_m : \neg root(t_j) \Rightarrow \\ \qquad (\exists t_i \in tc_m : parent(t_i, t_j) \wedge (t_j \, \mathcal{BD} \, t_i)) \end{array}$$

6. *Each transaction t_i of a transaction closure is connected to one of its subtransaction t_j by* exactly one *termination dependency:*

   $$\begin{array}{l} \forall tc_m : \forall t_i \in tc_m : \exists t_j \in tc_m : parent(t_i, t_j) \Rightarrow \\ \quad ((vital\_dep(t_i, t_j) \wedge \neg vital(t_i, t_j) \wedge \neg dep(t_i, t_j) \\ \qquad \wedge \neg exc(t_i, t_j) \wedge \neg indep(t_i, t_j)) \vee \\ \quad (\neg vital\_dep(t_i, t_j) \wedge vital(t_i, t_j) \wedge \neg dep(t_i, t_j) \\ \qquad \wedge \neg exc(t_i, t_j) \wedge \neg indep(t_i, t_j) \vee \\ \quad (\neg vital\_dep(t_i, t_j) \wedge \neg vital(t_i, t_j) \wedge dep(t_i, t_j) \\ \qquad \wedge \neg exc(t_i, t_j) \wedge \neg indep(t_i, t_j) \vee \\ \quad (\neg vital\_dep(t_i, t_j) \wedge \neg vital(t_i, t_j) \wedge \neg dep(t_i, t_j) \\ \qquad \wedge exc(t_i, t_j) \wedge \neg indep(t_i, t_j) \vee \\ \quad (\neg vital\_dep(t_i, t_j) \wedge \neg vital(t_i, t_j) \wedge \neg dep(t_i, t_j) \\ \qquad \wedge \neg exc(t_i, t_j) \wedge indep(t_i, t_j))) \end{array}$$

   For the sake of readability, we have disregarded the combined dependencies, e.g. $vital_d(t_i, t_j)$, and $vital_{\blacktriangleright}(t_i, t_j)$. Obviously, two transactions of a transaction closure may be interrelated by one of these combined dependencies.

7. *The execution order dependency between transactions is acyclic:* $\forall tc_m : \nexists t_i \in tc_m : seq(t_i, t_i)$

[3]The symbol $\exists!$ stands for "it exists exactly one".

7 Application of Transaction Closures

The following example illustrates the application of transaction closures. In particular this example shall show how to deal with transitive dependencies in transaction closure. The transaction closure in our example can be considered as a workflow with special dependencies among the different transactions. Especially, there are transactions which are executed outside the scope of the initiating transaction.

Example 26 *Let t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, and t_9 be transactions of a transaction closure with the root transaction t_1. The transaction t_2, t_3, and t_4 are subtransactions of the root transaction and connected by the following dependencies:*

$$vital\_dep(t_1, t_2) \wedge dep(t_1, t_3) \wedge vital_{\blacktriangleright}(t_1, t_4)$$

Furthermore, t_5 and t_6 are subtransactions of transaction t_2, t_7 is a subtransaction of transaction t_3, and t_8 and t_9 are subtransactions of transaction t_4. These subtransactions are connected to their parent transactions by the following dependencies:

$$\begin{array}{l} vital(t_2, t_5) \wedge dep(t_2, t_6) \wedge exc(t_3, t_7) \wedge \\ \qquad indep(t_4, t_8) \wedge vital\_dep_{\blacklozenge}(t_4, t_9) \end{array}$$

The transactions t_4, t_6, and t_9 are compensatable. Furthermore, execution order dependencies are defined among the transactions of the closure. In the following we state the sequential dependencies:

$$seq\_c(t_2, t_4) \wedge seq\_c(t_4, t_3) \wedge seq\_a(t_6, t_8)$$

The transaction t_4 is executed after the commit of transaction t_2 and transaction t_3 after the commit of t_4. Moreover, transaction t_8 is only executed after the abortion of transaction t_6. All other transactions are executed in parallel.

Our example transaction closure is illustrated in Figure 2. The arrows between transactions denote the direction of the abort dependencies like in Figure 1. Additionally to the symbols used in Figure 1, we now introduce the following symbols for the dependencies exclusive *and* independent*:*

$$\begin{array}{rcl} exc(t_i, t_j) & \textit{corresponds to} & t_i \nleftrightarrow t_j \\ indep(t_i, t_j) & \textit{corresponds to} & t_i \,—\, t_j \end{array}$$

From our dependency definitions and Theorems 6 and 23 we can now derive the transitive dependencies in the underlying transaction closure. We are also able to investigate

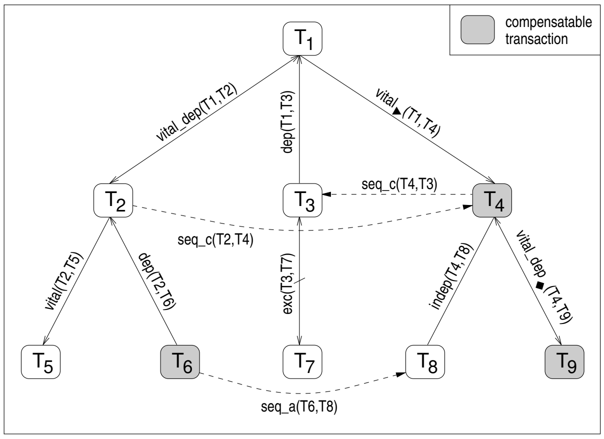

Figure 2. Example of a Transaction Closure

the influence of the abortion of a certain transaction on the whole closure. In the following we discuss some interesting cases. We start with the consideration of transactions t_2, t_5, and t_6. We know that t_2 is vital *for t_5 and* dependent *on t_6. From this basic dependencies we can derive that t_5 is* transitively dependent *on transaction t_6:*

$$\begin{aligned} & vital(t_2, t_5) \wedge dep(t_2, t_6) \\ \equiv\ & dep(t_5, t_2) \wedge dep(t_2, t_6) \quad \Longrightarrow \quad dep(t_5, t_6) \end{aligned}$$

Hence, the abortion of transaction t_6 leads to the abortion of the parent t_2 and the sibling t_5. In contrast, the abortion of t_5 has no influence on the other transactions. Concerning the whole transaction closure, an abortion of t_6 leads to an abortion of all other transactions except t_3, t_7, and t_8.

Transaction t_8 is independent *of its initiating transaction and, thus, it is* transitively independent *of all other transactions of the transaction closure. From this follows that an abortion of another transaction of the closure has no influence on t_8 and that the abortion of t_8 is without any effects on the other transactions. For example, the transaction t_8 may continue executing after the termination of its parent transaction t_4. Thus, transaction t_8 may leave the scope of its parent transaction.*

Transaction t_7 is exclusive *for its parent t_3. Consequently, t_7 is* transitively exclusive *for the transactions t_1, t_2, and t_5 and* transitively exclusive$_\blacklozenge$ *for the transactions t_4 and t_9 which are compensatable:*

$$dep(t_1, t_3) \wedge exc(t_3, t_7) \quad \Longrightarrow \quad exc(t_1, t_7)$$

$$\begin{aligned} & vital\_dep(t_1, t_2) \wedge exc(t_1, t_7) \\ \equiv\ & vital\_dep(t_2, t_1) \wedge exc(t_1, t_7) \quad \Longrightarrow \quad exc(t_2, t_7) \end{aligned}$$

$$\begin{aligned} & vital(t_2, t_5) \wedge exc(t_2, t_7) \\ \equiv\ & dep(t_5, t_2) \wedge exc(t_2, t_7) \quad \Longrightarrow \quad exc(t_5, t_7) \end{aligned}$$

$$\begin{aligned} & exc(t_1, t_7) \wedge vital_\blacktriangleright(t_1, t_4) \\ \equiv\ & exc(t_7, t_1) \wedge vital_\blacktriangleright(t_1, t_4) \quad \Longrightarrow \quad exc_\blacklozenge(t_7, t_4) \end{aligned}$$

$$exc_\blacklozenge(t_7, t_4) \wedge vital\_dep_\blacklozenge(t_4, t_9) \quad \Longrightarrow \quad exc_\blacklozenge(t_7, t_9)$$

Transaction t_8 is independent *of transaction t_7 for the reason discussed above. Furthermore, the transaction t_6 is* transitively independent *of t_7. The dependency between the transactions t_6 and t_2 is vital$_\blacktriangleright$ because transaction t_6 is compensatable (see Theorem 23). Due to the vital dependency, transaction t_6 may commit and t_2 abort. On the other hand, the exclusive dependency between t_2 and t_7 allows that t_2 aborts and transaction t_7 commits. In this case, both transactions t_6 and t_7 commit. Thus, the transitive dependency between t_6 and t_7 cannot be exclusive.*

$$exc_\blacklozenge(t_7, t_4) \wedge indep(t_4, t_8) \quad \Longrightarrow \quad indep(t_7, t_8)$$

$$\begin{aligned} & dep(t_2, t_6) \wedge exc(t_2, t_7) \\ \equiv\ & vital_\blacktriangleright(t_6, t_2) \wedge exc(t_2, t_7) \quad \Longrightarrow \quad indep(t_6, t_7) \end{aligned}$$

The subtransactions t_2, t_3, and t_4 of the root transaction t_1 are connected by the following transitive dependencies:

$$\begin{aligned} & vital\_dep(t_1, t_2) \wedge vital_\blacktriangleright(t_1, t_4) \\ \equiv\ & vital\_dep(t_2, t_1) \wedge vital_\blacktriangleright(t_1, t_4) \quad \Longrightarrow \quad vital_\blacktriangleright(t_2, t_4) \\ & vital\_dep(t_1, t_2) \wedge dep(t_1, t_3) \\ \equiv\ & vital\_dep(t_2, t_1) \wedge dep(t_1, t_3) \quad \Longrightarrow \quad dep(t_2, t_3) \\ & dep(t_1, t_3) \wedge vital_\blacktriangleright(t_1, t_4) \\ \equiv\ & vital(t_3, t_1) \wedge vital_\blacktriangleright(t_1, t_4) \quad \Longrightarrow \quad vital_\blacktriangleright(t_3, t_4) \end{aligned}$$

Finally, we show the derivation of a transitive execution dependency. Due to the fact that transaction t_4 is sequential-commit dependent *on transaction t_2 whereas transaction t_3 is* sequential-commit dependent *on transaction t_4, t_2 is* transitively sequential-commit dependent *on t_3.*

$$seq\_c(t_2, t_4) \wedge seq\_c(t_4, t_3) \Longrightarrow seq\_c(t_2, t_3)$$

Example 26 showed that the abortion of a transaction may lead to the abortion of parts of the closure. Dependencies between two arbitrary transactions may be complex and sometimes not obvious. The influence of transaction abortion on a transaction closure can be simulated by the transitive dependencies. This may help also to detect unnecessary parts of a transaction closure definition. In the following example we illustrate a dependency specification which is contradictory.

Example 27 *Let t_i and t_j be two transactions which are connected by an* exclusive *dependency and suppose that transaction t_i is* sequential-commit dependent *on t_j:*

$$exc(t_i, t_j) \wedge seq\_c(t_i, t_j)$$

The sequential-commit dependency requires that the initiating of transaction t_j follows the commitment of transaction t_i. However, the combination of this execution dependency with the exclusive dependency implies that transaction t_i always has to be aborted. Due to the exclusive dependency, the commit of t_j leads to the abort of t_i. In case t_j aborts, transaction t_i is not started because of the *sequential-commit dependency. In other words, the execution of transaction t_i makes no sense. Consequently, either the specification of such a transaction is superfluous or a failure happens during the transaction closure design process. Such failures are hints for the transaction designer that there are dependencies which are incorrect according to the real-world applications semantics.*

8 Conclusions and Outlook

Nested transaction structures do not provide an adequate platform for various complex applications, e.g. for applications which are based on activity or workflow models. In this paper, we have presented a generalized framework for describing and classifying related transactions in a uniform way, independent of how complex they are interrelated. The concept of transaction closures extends the concept of nested transactions, for example, allowing detached transactions in such transaction closures. *Detached* transactions [2] are modeled as subtransactions which are independent of the initiating transaction. Such subtransactions may leave the scope of the initiating transaction. By putting further constraints on such kinds of transactions we can also model different types of detached transactions, e.g. *detached but causally dependent* transactions. This extensions are especially relevant for applications of active databases, federated databases, and mobile computing.

In particular, our framework supports the automatic derivation of transitive dependencies. This issue is important to get a grasp of the entire semantics of a complex application. By this way, it is possible to conclude how two (arbitrary) transactions are interrelated. For instance, the application designer can estimate which parts (transactions) of the application are concerned by an abortion of a certain transaction. Thus, failures or redundancies in the application specification can be detected during the design phase. This would help to develop less failure-prone applications.

Our future work will focus on the aspect of the enforcement and implementation of transaction dependencies. Here, we will attempt to adopt the methods proposed in [7, 9] to our framework and provide some extensions to capture the transitive properties of transaction dependencies.

Acknowledgements: We are grateful to Stefan Conrad for useful remarks.

References

[1] P. C. Attie, M. Singh, A. Sheth, and M. Rusinkiewicz. Specifying and Enforcing Intertask Dependencies. In S. Baker, R. Agrawal, and D. Bell, eds., *Proc. 19th Int. Conf. on Very Large Data Bases*, pp. 134–145, Morgan Kaufmann, 1993.

[2] A. P. Buchmann. Active Object Systems. In A. Dogac, M. T. Özsu, A. Biliris, and T. Sellis, eds., *Advances in Object-Oriented Database Systems*, pp. 201–224, Springer, 1994.

[3] P. K. Chrysanthis and K. Ramamritham. A Formalism for Extended Transaction Models. In G. M. Lohmann, A. Sernadas, and R. Camps, eds., *Proc. 17th Int. Conf. on Very Large Data Bases*, pp. 103–112, Morgan Kaufmann, 1991.

[4] P. K. Chrysanthis and K. Ramamritham. Synthesis of Extended Transaction Models Using ACTA. *ACM Transaction on Database Systems*, 19(3):450–491, September 1994.

[5] U. Dayal, M. Hsu, and R. Ladin. A Transaction Model for Long-Running Activities. In G. M. Lohmann, A. Sernadas, and R. Camps, eds., *Proc. 17th Int. Conf. on Very Large Data Bases*, pp. 113–122, Morgan Kaufmann, 1991.

[6] A. K. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A Multidatabase Transaction Model for InterBase. In D. McLeod, R. Sacks-Davis, and H.-J. Schek, eds., *Proc. 16th Int. Conf. on Very Large Data Bases*, pp. 507–518, Morgan Kaufmann, 1990.

[7] D. Georgakopoulos, M. Hornick, and P. Krychniak. An Environment for the Specification and Management of Extended Transactions in DOMS. In H.-J. Schek, A. P. Sheth, and B. D. Czejdo, eds., *Proc. 3rd Int. Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems*, pp. 253–257, IEEE Computer Society Press, 1993.

[8] D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, April 1995.

[9] A. Geppert and K. R. Dittrich. Rule-Based Implementation of Transaction Model Specifications. In N. W. Paton and M. H. Williams, eds., *Rules in Database Systems, Proc. 1st Int. Workshop*, pp. 127–142, Springer, 1994.

[10] M. Hsu, R. Ladin, and D. R. McCarthy. An Execution Model For Active Data Base Management Systems. In C. Beeri, J. W. Schmidt, and U. Dayal, eds., *Proc. 3rd Int. Conf. on Data and Knowledge Bases: Improving Usability and Responsiveness*, pp. 171–179, Morgan Kaufmann, 1988.

[11] M. U. Kamath and K. Ramamritham. Correctness Issues in Workflow Management. *Distributed Systems Engineering*, 3(4), December 1996.

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

[13] K. Schwarz, C. Türker, and G. Saake. Analyzing and Formalizing Dependencies in Generalized Transaction Structures. In *Proc. of the Int. Workshop on Issues and Applications of Database Technology (IADT'98), July 6-9, 1998, Berlin, Germany*, 1998. *To appear*.

[14] K. Schwarz, C. Türker, and G. Saake. Derived Transaction Termination Dependencies: An Algorithm for Computing Transitivity Rules. Preprint 7, Fakultät für Informatik, Universität Magdeburg, February 1998.

[15] H. Wächter and A. Reuter. The ConTract Model. In A. K. Elmagarmid, ed., *Database Transaction Models for Advanced Applications*, pp. 219–263, Morgan Kaufmann, 1992.