T. Özsu, A. Dogac, Ö. Ulusoy (eds.), Issues and Applications of Database Technology (IADT'98), Proc. of the 3rd World Conf. on Integrated Design and Process Technology, July 6-9, 1998, Berlin, Germany, pages 55-62. Society for Design and Process Science, IDPT-Vol. 2, 1998.

ANALYZING AND FORMALIZING DEPENDENCIES IN GENERALIZED TRANSACTION STRUCTURES*

Kerstin Schwarz Can Türker Gunter Saake

Otto-von-Guericke-Universität Magdeburg
Postfach 4120, 39016 Magdeburg, Germany
E-mail: {schwarz|tuerker|saake}@iti.cs.uni-magdeburg.de

ABSTRACT

Modern information systems require advanced transaction models providing means to describe complex activities such as transactional workflows. Complex activities consist of sets of transactions which are interrelated, i.e., there are dependencies among several transactions. We analyze different kinds of dependencies and present an extended formal framework (based on ACTA) for describing advanced transaction models. We introduce the notion of transaction closures as a generalization of nested transactions which captures different orthogonal dependencies such as abort and object visibility dependencies, and explicitly show how these dependencies work together. The formal framework is demonstrated by formalizing detached transactions known from the area of active database systems.

1 Introduction

Advanced transaction models are the basis for realizing complex activities in information systems. Traditionally, such models base on the idea of nesting transactions. Nested transactions (Mos85) consist of smaller transactions called *child* transactions. The initiating transaction is called *parent*. A child transaction may also consist of child transactions. Depending on the different variations of the nested transaction model, child transactions can be executed sequentially or parallel, and the nesting of the transactions can be closed or open (WS92). However, all transactions have to be terminated before the parent transaction can be committed. This follows immediately from the term 'nesting' which means to include something.

In recent years, especially with the introduction of active databases (DHW95), there is an increasing demand for transaction models which provide a more general framework in order to support more flexible transactions realizing complex activities. For instance, detached transactions with various modes are required (Buc94). A *detached* transaction is initiated by another transaction and is executed as a separate transaction. In comparison to classical child transactions, detached transactions may leave the execution boundaries of the initiating transaction. Depending on the detached mode, the abortion of the initiating transaction may not have any influence on a detached transaction.

The DOM transaction model (BÖH+92) goes in this direction by providing different kinds of transactions such as *vital*, *non-vital*, *contingency*, and *compensating* transactions. A further step towards a more general transactional execution framework are the activity model (DHL91), the ConTract model (WR92), and (transactional) workflow models (RS95). All these models have in common that they support the specification of a certain control flow among transactions.

We present a general formal framework for describing advanced transaction models. For that, we extend the ACTA formalism (CR94) concerning aspects of activity models. One extension is the notion of *transaction closures* generalizing nested transactions. A transaction closure contains all transactions initiated (transitively) by one (root) transaction. In contrast to nested transactions, a transaction closure may consist of transactions[1] which terminate after the termination of their parents. In other words, a parent can terminate without waiting for the termination of the child.

There may be different kinds of dependencies among transactions of a transaction closure. We analyze an orthogonal set of dependencies. First, we consider abort dependencies in both directions, from parent to child transactions and vice versa. Furthermore, we investigate how results of transactions are made visible to other transactions. Another interesting issue concerns execution order dependencies being used to describe the control flow among transactions.

The paper is organized as follows: Section 2 defines basic properties of advanced transactions. In Section 3 we analyze abort dependencies among transactions. Section 4 discusses object visibility dependencies whereas Section 5 deals with execution order dependencies. In Section 6 the notion of transaction closures is introduced and different dependencies are combined. The application of our formal framework to detached transactions is demonstrated in Section 7.

2 Foundations

A *transaction* is an execution unit consisting of a set of operations. A transaction t_i is started by invoking the transaction management primitive begin (b_{t_i}) and

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

[1] We explicitly do not use the term subtransaction because traditionally subtransactions are executed in the scope of the parent. From our point of view, a parent initiates child transactions without any requirements on the termination of these transactions.

is terminated either by commit (c_{t_i}) or abort (a_{t_i}). A transaction invokes operations, termed as *object events*, to access and manipulate the state of database objects. The object event $p_{t_i}[ob]$ refers to the invocation of an operation p on an object ob by a transaction t_i. The effects of an object event of t_i are made persistent by using the operation $\mathsf{c}[p_{t_i}[ob]]$ and are obliterated by the operation $\mathsf{a}[p_{t_i}[ob]]$. The access set AS_{t_i} contains all operations $p[ob]$ for which transaction t_i is responsible. The commit of t_i commits all its operations.

A *history* of a concurrent execution of a set of transactions T is a partial order of all events associated with the transactions in T. The complete history H contains only terminated transactions, the current (incomplete) history is termed as H_{ct}. The occurrence of an event in a history may be constrained as follows:

(1) *An event ϵ' may occur only after another event ϵ in a history* H: $\epsilon \to \epsilon'$ is true iff ϵ precedes ϵ' in H.

(2) *An event ϵ may occur only if a certain condition is true*: $(\epsilon \in \mathsf{H}) \Rightarrow Cond_H$ specifies ϵ can belong to H only if $Cond_H$ is satisfied. $Cond_H$ is a predicate involving the events in H, e.g. $(\epsilon' \to \epsilon)$. It can be formulated using well-known logical connectives, e.g., $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \forall, \exists$.

(3) *A condition requires the occurrence of an event*: $Cond_H \Rightarrow (\epsilon \in \mathsf{H})$ specifies that ϵ should be in history H if $Cond_H$ holds.

A transaction must be in one of the following states:

$$\begin{aligned}
\mathsf{active}(t_i) &:\Leftrightarrow \mathsf{b}_{t_i} \in \mathsf{H}_{ct} \wedge \mathsf{c}_{t_i} \notin \mathsf{H}_{ct} \wedge \mathsf{a}_{t_i} \notin \mathsf{H}_{ct} \\
\mathsf{committed}(t_i) &:\Leftrightarrow \mathsf{c}_{t_i} \in \mathsf{H}_{ct} \\
\mathsf{aborted}(t_i) &:\Leftrightarrow \mathsf{a}_{t_i} \in \mathsf{H}_{ct}
\end{aligned}$$

The effects of transactions on other transactions are described by dependencies being constraints on possible histories. In the sequel, we recall some fundamental dependencies described in more detail in (CR94). Let t_i and t_j be transactions and H be a finite history:

Begin Dependency $(t_i\, \mathcal{BD}\, t_j)$**.** Transaction t_i can only be initiated if t_j is already initiated: $\mathsf{b}_{t_i} \in \mathsf{H} \Rightarrow (\mathsf{b}_{t_j} \to \mathsf{b}_{t_i})$

Commit Dependency $(t_i\, \mathcal{CD}\, t_j)$**.** If t_i and t_j commit, then the commit of t_j must precede the commit of t_i: $\mathsf{c}_{t_i} \in \mathsf{H} \Rightarrow (\mathsf{c}_{t_j} \in \mathsf{H} \Rightarrow (\mathsf{c}_{t_j} \to \mathsf{c}_{t_i}))$

Abort Dependency $(t_i\, \mathcal{AD}\, t_j)$**.** If t_j aborts, t_i has to abort, too: $\mathsf{a}_{t_j} \in \mathsf{H} \Rightarrow \mathsf{a}_{t_i} \in \mathsf{H}$

Weak-Abort Dependency $(t_i\, \mathcal{WD}\, t_j)$**.** If t_i commits and t_j aborts, the commit of t_i precedes the abortion of t_j: $\mathsf{a}_{t_j} \in \mathsf{H} \Rightarrow ((\mathsf{c}_{t_i} \to \mathsf{a}_{t_j}) \vee (\mathsf{a}_{t_i} \in \mathsf{H}))$

Serial Dependency $(t_i\, \mathcal{SD}\, t_j)$**.** Transaction t_i cannot begin executing until t_j either commits or aborts: $\mathsf{b}_{t_i} \in \mathsf{H} \Rightarrow ((\mathsf{c}_{t_j} \to \mathsf{b}_{t_i}) \vee (\mathsf{a}_{t_j} \to \mathsf{b}_{t_i}))$

Begin-on-Abort Dependency $(t_i\, \mathcal{BAD}\, t_j)$**.** Transaction t_i cannot begin its execution until t_j aborts: $\mathsf{b}_{t_i} \in \mathsf{H} \Rightarrow (\mathsf{a}_{t_j} \to \mathsf{b}_{t_i})$

Begin-on-Commit Dependency $(t_i\, \mathcal{BCD}\, t_j)$**.** Transaction t_i cannot begin executing until t_j commits: $(\mathsf{b}_{t_i} \in \mathsf{H}) \Rightarrow (\mathsf{c}_{t_j} \to \mathsf{b}_{t_i})$

Additionally to the dependencies from (CR94) stated above, we introduce the following dependencies:

Force-Begin-on-Abort Dependency $(t_i\, \mathcal{FBAD}\, t_j)$**.** If t_j aborts, t_i has to begin: $\mathsf{a}_{t_j} \in \mathsf{H} \Rightarrow (\mathsf{a}_{t_j} \to \mathsf{b}_{t_i})$

Commit-on-Termination Dependency $(t_i\, \mathcal{CTD}\, t_j)$**.** Transaction t_i cannot commit until t_j either commits or aborts: $\mathsf{c}_{t_i} \in \mathsf{H} \Rightarrow ((\mathsf{c}_{t_j} \to \mathsf{c}_{t_i}) \vee (\mathsf{a}_{t_j} \to \mathsf{c}_{t_i}))$

Begin-before-Abort Dependency $(t_i\, \mathcal{BBAD}\, t_j)$**.** Transaction t_i cannot abort until t_j starts its execution: $\mathsf{a}_{t_i} \in \mathsf{H} \Rightarrow (\mathsf{b}_{t_j} \to \mathsf{a}_{t_i})$

Begin-before-Commit Dependency $(t_i\, \mathcal{BBCD}\, t_j)$**.** Transaction t_i cannot commit until transaction t_j starts executing: $\mathsf{c}_{t_i} \in \mathsf{H} \Rightarrow (\mathsf{b}_{t_j} \to \mathsf{c}_{t_i})$

Advanced transactions are structured, i.e., they consist of sets of transactions which are interrelated.

Definition 2.1 *The following predicates describe general relationships between a transaction and its creator:*

$$\begin{aligned}
\mathsf{root}(t_j) &:= \{t_j \textit{ has no parent}\} \\
\mathsf{parent}(t_i, t_j) &:= \{t_i \textit{ is parent of } t_j\} \\
\mathsf{superior}(t_i, t_j) &:= (\mathsf{parent}(t_i, t_j) \vee \\
&\qquad (\exists t_k : \mathsf{superior}(t_i, t_k) \wedge \mathsf{parent}(t_k, t_j)))
\end{aligned}$$

Definition 2.2 *Suppose $\mathcal{ST}$ denotes the set of structured transactions:*

- *Each structured transaction has exactly one root:* $\forall st_k \in \mathcal{ST} : \dot{\exists} t_i \in st_k : \mathsf{root}(t_i)$
- *Each non-root transaction has exactly one parent and the begin of the child must follow the begin of the parent:* $\forall st_k \in \mathcal{ST} : \forall t_j \in st_k : \neg\mathsf{root}(t_j) \Rightarrow (\dot{\exists} t_i \in st_k : \mathsf{parent}(t_i, t_j) \wedge (t_j\, \mathcal{BD}\, t_i))$
- *Each transaction structure is acyclic:* $\forall st_k \in \mathcal{ST} : \nexists t_i \in st_k : \mathsf{superior}(t_i, t_i)$

3 Abort Dependencies

Fundamental dependencies in transaction structures are *abort dependencies* which state whether the abortion of a transaction has an influence on another transaction. In the literature, e.g. (CR94), the abort dependencies in nested transactions are well investigated. However, due to the basic assumption of nested transactions that a child transaction may not leave the scope of the parent, the abortion of a parent implies the abortion of all its child transactions. Consequently *different* abort dependencies are only considered in one direction (from a child to its parent). In contrast, we consider general transaction structures and thus distinguish between different *upwards* and *downwards* abort dependencies, i.e., different possible effects of a transaction abortion on superiors as well as on child transactions.

3.1 Influences of Abortions on Superiors

First, we consider the effects of an abortion of a transaction on its parent and other superiors. We define four kinds of upwards abort dependencies: *vital*, *weak-vital*, *weak-non-vital*, and *non-vital*.

Definition 3.1 (Vital) *A transaction t_j is vital for another transaction t_i iff t_i is (transitively) abort as well as commit dependent on t_j:*

$$\mathsf{vital}(t_i,t_j) :\Leftrightarrow (\mathsf{parent}(t_i,t_j) \wedge (t_i\,\mathcal{AD}\,t_j) \wedge (t_i\,\mathcal{CD}\,t_j)) \vee (\exists t_k : \mathsf{vital}(t_i,t_k) \wedge \mathsf{vital}(t_k,t_j))$$

Thus, an abortion of a *vital* transaction t_j leads to the abortion of all superiors for which t_j is vital. In other words, a transaction is only allowed to commit, if all its vital child transactions are committed.

Upwards abort dependencies have an influence on the commit of the parent and on the abortion of a child. Table 1 illustrates that a parent t_i of a vital child t_j may only commit if t_j is committed. Due to the fact, that an abortion of a vital child leads to the abortion of the parent, the case, in which t_i wants to commit and t_j is already aborted, cannot occur. If t_j wants to abort, we have to consider the state of t_i. In this case, t_i cannot be committed until the vital child t_j is committed.

$\mathsf{vital}(t_i,t_j)$	t_i wants to commit
$\mathsf{active}(t_j)$	t_i must wait for t_j's termination and may only commit if t_j commits
$\mathsf{committed}(t_j)$	t_i may commit
$\mathsf{aborted}(t_j)$	this case cannot occur

$\mathsf{vital}(t_i,t_j)$	t_j wants to abort
$\mathsf{active}(t_i)$	t_j may abort and t_i must be aborted
$\mathsf{committed}(t_i)$	this case cannot occur
$\mathsf{aborted}(t_i)$	t_j may abort (no effects on t_i)

Table 1 Vital Dependency

Our second upwards abort dependency bases on the notion of a *contingency* transaction.

Definition 3.2 (Contingency) *Transaction t_c is a contingency transaction of transaction t_j iff t_c is semantically equivalent*[2] *to t_j, t_c is force-begin-on-abort dependent on t_j, the common parent t_i of t_c and t_j is commit dependent on t_c, and t_i is abort dependent on t_c or there exists a contingency transaction t_m for t_c:*

$$\mathsf{cont}(t_c,t_j) :\Leftrightarrow \mathsf{sem\text{-}equiv}(t_c,t_j) \wedge (t_c\,\mathcal{FBAD}\,t_j) \wedge (\exists t_i : \mathsf{parent}(t_i,t_j) \wedge \mathsf{parent}(t_i,t_c) \wedge (t_i\,\mathcal{CD}\,t_c) \wedge ((t_i\,\mathcal{AD}\,t_c) \vee (\exists t_m : \mathsf{cont}(t_m,t_c))))$$

A contingency transaction is executed as an alternative to transaction t_j in case t_j aborts and may have itself a contingency transaction. Thus, a transaction t_c is termed as an alternative to t_j iff t_c is (transitively) a contingency transaction of t_j:

$$\mathsf{alter}(t_c,t_j) :\Leftrightarrow \mathsf{cont}(t_c,t_j) \vee (\exists t_k : \mathsf{alter}(t_c,t_k) \wedge \mathsf{cont}(t_k,t_j))$$

Definition 3.3 (Weak-Vital) *A transaction t_j is weak-vital for another transaction t_i iff t_i is (transitively) commit dependent on t_j, begin-before-commit dependent on the contingency transaction t_c of t_j in case t_j aborts*[3]*, and there is an alternative t_m of t_j which is vital to terminate the execution of t_j's alternatives:*

$$\mathsf{weak\text{-}vital}(t_i,t_j) :\Leftrightarrow (\mathsf{parent}(t_i,t_j) \wedge (t_i\,\mathcal{CD}\,t_j) \wedge (\dot{\exists} t_c : (\mathsf{cont}(t_c,t_j) \wedge abort_{t_j} \in H) \Rightarrow (t_i\,\mathcal{BBCD}\,t_c)) \wedge (\exists t_m : \mathsf{alter}(t_m,t_j) \wedge \mathsf{vital}(t_i,t_m))) \vee (\exists t_k : (\mathsf{weak\text{-}vital}(t_i,t_k) \wedge \mathsf{weak\text{-}vital}(t_k,t_j)) \vee (\mathsf{weak\text{-}vital}(t_i,t_k) \wedge \mathsf{vital}(t_k,t_j)) \vee (\mathsf{vital}(t_i,t_k) \wedge \mathsf{weak\text{-}vital}(t_k,t_j)))$$

Table 2 shows the case in which a parent t_i of a weak-vital transaction t_j wants to commit. Transaction t_i may only commit if t_j or an alternative of t_j commits. Hence, a weak-vital child can abort without forcing the parent to abort, if there exists an alternative transaction which commits. In case t_j wants to abort, t_j may

$\mathsf{weak\text{-}vital}(t_i,t_j)$	t_i wants to commit
$\mathsf{active}(t_j)$	t_i must wait for t_j's termination and may only commit if t_j or an alternative commits
$\mathsf{committed}(t_j)$	t_i may commit
$\mathsf{aborted}(t_j)$	t_i may commit

$\mathsf{weak\text{-}vital}(t_i,t_j)$	t_j wants to abort
$\mathsf{active}(t_i)$	t_j may abort but $\mathsf{cont}(t_c,t_j)$ must be initiated
$\mathsf{committed}(t_i)$	this case cannot occur
$\mathsf{aborted}(t_i)$	t_j may abort (no effects on t_i)

Table 2 Weak-Vital Dependency

abort but a contingency transaction t_c of t_j must be executed afterwards. However, t_i cannot be committed because t_i has to wait for the termination of t_j.

In contrast, the abortion of a *non-vital* child has no influence on the superior, i.e., the parent may commit even if a child aborts. Here, we distinguish two kinds of non-vital dependencies. A transaction t_j is called *weak-non-vital* if the termination of t_j has to precede the commitment of the superior. Please note that the notion of weak-non-vital we used here is usually termed as non-vital in the literature (see e.g. (Mos85; Elm92)). From our point of view, a transaction is non-vital if the abortion of this transaction has no influence on commitment of the parent and additionally the parent has not to wait for the termination of a non-vital child. This distinction in terminology is necessary because we are working with more general transaction structures.

Definition 3.4 (Weak-Non-Vital) *A transaction t_j is weak-non-vital for another transaction t_i iff t_i is (transitively) commit-on-termination dependent on t_j:*

$$\mathsf{weak\text{-}non\text{-}vital}(t_i,t_j) :\Leftrightarrow (\mathsf{parent}(t_i,t_j) \wedge (t_i\,\mathcal{CTD}\,t_j)) \vee (\exists t_k : (\mathsf{weak\text{-}non\text{-}vital}(t_i,t_k) \wedge (\mathsf{vital}(t_k,t_j) \vee \mathsf{weak\text{-}vital}(t_k,t_j))) \vee ((\mathsf{vital}(t_i,t_k) \vee \mathsf{weak\text{-}vital}(t_i,t_k)) \wedge \mathsf{weak\text{-}non\text{-}vital}(t_k,t_j)) \vee (\mathsf{weak\text{-}non\text{-}vital}(t_i,t_k) \wedge \mathsf{weak\text{-}non\text{-}vital}(t_k,t_j)))$$

Table 3 illustrates the commitment of a parent t_i of a weak-non-vital child t_j. Transaction t_i may commit if t_j is already terminated. Thus, the case that t_i is committed and the child t_j wants to abort cannot occur.

[2] The transaction designer defines which transactions are semantically equivalent ($\mathsf{sem\text{-}equiv}(t_c,t_j)$).

[3] This is necessary to guarantee that the contingency transaction t_c do not leave the scope of its parent t_i.

weak-non-vital(t_i, t_j)	t_i wants to commit
active(t_j)	t_i must wait for t_j's termination
committed(t_j)	t_i may commit
aborted(t_j)	t_i may commit

weak-non-vital(t_i, t_j)	t_j wants to abort
active(t_i)	t_j may abort
committed(t_i)	this case cannot occur
aborted(t_i)	t_j may abort (no effects on t_i)

Table 3 Weak-Non-Vital Dependency

The abortion of a *non-vital* child has no influence on the parent. The parent may continue even if a non-vital child aborts and can commit without waiting for the termination of its non-vital child transactions.

Definition 3.5 (Non-Vital) *Transaction t_j is non-vital for another transaction t_i iff the abortion of t_j has no effects on the termination of t_i.*

3.2 Effects of Abortions on Child Transactions

Investigating the abort of a child and the influences on their superiors leads to the question in which way an abort of a superior may effect transactions. We define the *dependent*, *weak-dependent* and *independent* types.

A *dependent* child aborts if the parent aborts and is only allowed to commit if the parent commits.

Definition 3.6 (Dependent) *A transaction t_j is dependent on another transaction t_i iff t_j is (transitively) abort dependent on t_i:*

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

Table 4 represents a parent t_i which may abort, if the dependent child t_j is active or aborted. The case that t_j is already committed is not valid, because the abort dependency $(t_j\, \mathcal{AD}\, t_i)$ has to be fulfilled. The child t_j must wait for the termination of t_i to commit because the abortion of t_i leads to the abortion of t_j.

dep(t_i, t_j)	t_i wants to abort
active(t_j)	t_i may abort and t_j must be aborted
committed(t_j)	this case cannot occur
aborted(t_j)	t_i may abort (no effects on t_j)

dep(t_i, t_j)	t_j wants to commit
active(t_i)	t_j must wait for t_i's termination
committed(t_i)	t_j may commit
aborted(t_i)	this case cannot occur

Table 4 Dependent Dependency

A *weak-dependent* transaction has to be aborted if a superior aborts. In contrast to a dependent transaction, a weak-dependent transaction is allowed to commit if its commitment precedes the termination of the parent.

Definition 3.7 (Weak-Dependent) *Transaction t_j is weak-dependent on another transaction t_i iff t_j is (transitively) weak-abort dependent on t_i:*

$$\begin{aligned}\mathsf{weak\text{-}dep}(t_i, t_j) :\Leftrightarrow\ & (\mathsf{parent}(t_i, t_j) \wedge (t_j\, \mathcal{WD}\, t_i)) \vee \\ & (\exists t_k : (\mathsf{weak\text{-}dep}(t_i, t_k) \wedge \mathsf{weak\text{-}dep}(t_k, t_j)) \vee \\ & (\mathsf{weak\text{-}dep}(t_i, t_k) \wedge \mathsf{dep}(t_k, t_j)) \vee \\ & (\mathsf{dep}(t_i, t_k) \wedge \mathsf{weak\text{-}dep}(t_i, t_k)))\end{aligned}$$

Table 5 illustrates that a parent t_i of a weak-dependent child t_j may abort independently of the state of t_j. In case t_j is already committed, then t_j has to be undone or compensated (if atomicity is required). On the other hand, t_j may commit while its superior t_i is active or committed. Transaction t_i cannot be aborted because its abortion leads to the abortion of the child t_j.

weak-dep(t_i, t_j)	t_i wants to abort
active(t_j)	t_i may abort and t_j must be aborted
committed(t_j)	t_i may abort but t_j must be undone
aborted(t_j)	t_i may abort (no effects on t_j)

weak-dep(t_i, t_j)	t_j wants to commit
active(t_i)	t_j may commit
committed(t_i)	t_j may commit
aborted(t_i)	this case cannot occur

Table 5 Weak-Dependent Dependency

An *independent* transaction, on the other hand, may continue even if its parent or another superior aborts. Thus, the commitment of an independent transaction is independent of an abortion of the superior.

Definition 3.8 (Independent) *Transaction t_j is independent of another transaction t_i iff the abortion of t_i has no effects on the termination of t_j.*

4 Object Visibility Dependencies

A second category of dependencies among transactions concerns the visibility of effects of transactions to other transactions. We have again to investigate two directions of object visibility dependencies: (1) visibility of the effects of a transaction on its child transactions and (2) visibility of the effects in the other direction.

4.1 Visibility of Effects on Child Transactions

Before considering different object visibility dependencies, we have to introduce the notion of the view of a transaction.

Definition 4.1 (View) *The view of a transaction t (denoted by view_t) specifies the state of objects visible to transaction t at a point in time.*

Definition 4.2 (Inheriting) *A transaction t_j is an inheriting transaction from the viewpoint of transaction t_i iff t_j is (transitively) inheriting the view of t_i:*

$$\begin{aligned}\mathsf{inher}(t_i, t_j) :\Leftrightarrow\ & (\mathsf{parent}(t_i, t_j) \wedge (\mathsf{view}_{t_j} = \mathsf{view}_{t_i} \cup \mathsf{AS}_{t_j})) \\ & \vee(\exists t_k : \mathsf{inher}(t_i, t_k) \wedge \mathsf{inher}(t_k, t_j))\end{aligned}$$

The formula above expresses that t_j sees, additionally to its own access set, everything what t_i sees. Here we used the term 'inheriting' to indicate that a transaction inherits the view of its parent and thus transitively the views of all inheriting superiors. In traditional nested transactions as introduced by Moss (Mos85) all child transactions are inheriting transactions.

Definition 4.3 (Non-inheriting) *A transaction t_j is a non-inheriting transaction from the viewpoint of* *transaction t_i iff t_j does not (transitively) inherits the view of t_i:*

$$\begin{aligned}\mathsf{non\text{-}inher}(t_i,t_j) :\Leftrightarrow\ & (\mathsf{parent}(t_i,t_j) \wedge \\ & (\mathsf{view}_{t_j} = \{p_t[ob] \mid \mathsf{committed})\} \cup \mathsf{AS}_{t_j})) \vee \\ & (\exists t_k : \mathsf{non\text{-}inher}(t_i,t_k) \vee \mathsf{non\text{-}inher}(t_k,t_j))\end{aligned}$$

A non-inheriting transaction sees after its creation only the effects of committed transactions. For example, detached transactions (Buc94) are non-inheriting.

4.2 Treatment of Results of Child Transactions

Now, we consider how the visibility of results of child transactions can be handled. We distinguish two cases: The effects are made visible (1) only to the parent or (2) to all other transactions. In the first case the effects of a transaction are *delegated* to the parent.

Definition 4.4 (Delegate) *Transaction t_j delegates the responsibility for committing or aborting its access set to transaction t_i by using the operation $\mathsf{d}_{t_j}[t_i]$. After $\mathsf{d}_{t_j}[t_i]$ the access set of t_i includes t_j's access set.*

Definition 4.5 (Delegating) *A transaction t_j is a delegating transaction from the viewpoint of transaction t_i if the access set of t_j is (transitively) delegated to t_i:*

$$\begin{aligned}\mathsf{del}(t_i,t_j) :\Leftrightarrow\ & (\mathsf{parent}(t_i,t_j) \wedge (\mathsf{c}_{t_j} \in \mathsf{H} \Leftrightarrow \mathsf{d}_{t_j}[t_i] \in \mathsf{H}) \wedge \\ & (\mathsf{d}_{t_j}[t_i] \in \mathsf{H} \Rightarrow (\mathsf{d}_{t_j}[t_i] \rightarrow \mathsf{c}_{t_i} \vee \mathsf{d}_{t_j}[t_i] \rightarrow \mathsf{a}_{t_i}))) \vee \\ & (\exists t_k : \mathsf{del}(t_i,t_k) \wedge \mathsf{del}(t_k,t_j))\end{aligned}$$

A delegating transaction t_j commits through delegation implying that with the commit of t_j the access set of t_j is made visible only to its parent t_i. As a consequence, t_i cannot terminate until t_j delegates its access set.

For performance reasons, often it is necessary that a child makes its effects visible to concurrent transactions and, thus, release all accessed and manipulated objects with the commit. This may happen before the termination of the parent.

Definition 4.6 (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:*

$$\begin{aligned}\mathsf{rel}(t_i,t_j) :\Leftrightarrow\ & (\mathsf{parent}(t_i,t_j) \wedge \\ & (\exists ob\ \exists p\ \mathsf{c}_{t_j}[p[ob]] \in \mathsf{H} \Rightarrow \mathsf{c}_{t_j} \in \mathsf{H}) \wedge \\ & (\mathsf{c}_{t_j} \in \mathsf{H} \Rightarrow (\forall p[ob] \in \mathsf{AS}_{t_j} \Rightarrow \mathsf{c}_{t_j}[p[ob]] \in \mathsf{H}))) \vee \\ & (\exists t_k : \mathsf{rel}(t_i,t_k) \vee \mathsf{rel}(t_k,t_j))\end{aligned}$$

5 Execution Order Dependencies

In general, there are execution order dependencies among transactions, e.g. the sequential execution of a contingency transaction after the abortion of another transaction. We distinguish the following execution order dependencies: *sequential* and *parallel*.

Definition 5.1 (Sequential) *A transaction t_j is sequential dependent on another transaction t_i iff t_j is (transitively) serial dependent on t_i. Analogously, the sequential-abort dependency and the sequential-commit dependency are defined:*

$$\begin{aligned}\mathsf{seq}(t_i,t_j) :\Leftrightarrow\ & (t_j\,\mathcal{SD}\,t_i) \vee (\exists t_k : \mathsf{seq}(t_i,t_k) \wedge \mathsf{seq}(t_k,t_j)) \\ \mathsf{seq\text{-}a}(t_i,t_j) :\Leftrightarrow\ & (t_j\,\mathcal{BAD}\,t_i) \vee \\ & (\exists t_k : \mathsf{seq\text{-}a}(t_i,t_k) \wedge \mathsf{seq}(t_k,t_j)) \\ \mathsf{seq\text{-}c}(t_i,t_j) :\Leftrightarrow\ & (t_j\,\mathcal{BCD}\,t_i) \vee \\ & (\exists t_k : \mathsf{seq\text{-}c}(t_i,t_k) \wedge \mathsf{seq}(t_k,t_j))\end{aligned}$$

Definition 5.2 (Parallel) *A transaction t_i is parallel to another transaction t_j iff t_j is not sequential dependent on t_i. Transaction t_i is parallel-abort dependent on t_j iff t_i is begin-before-abort dependent on t_j and parallel-commit dependent on t_j iff t_i is begin-before-commit dependent on t_j:*

$$\begin{aligned}\mathsf{par}(t_i,t_j) :\Leftrightarrow\ & \neg\mathsf{seq}(t_i,t_j) \\ \mathsf{par\text{-}a}(t_i,t_j) :\Leftrightarrow\ & (t_i\,\mathcal{BBAD}\,t_j) \vee \\ & (\exists t_k : \mathsf{par\text{-}a}(t_i,t_k) \wedge \mathsf{seq}(t_k,t_j)) \\ \mathsf{par\text{-}c}(t_i,t_j) :\Leftrightarrow\ & (t_i\,\mathcal{BBCD}\,t_j) \vee \\ & (\exists t_k : \mathsf{par\text{-}c}(t_i,t_k) \wedge \mathsf{seq}(t_k,t_j))\end{aligned}$$

6 Transaction Closures and Dependency Combinations

A transaction closure is a generalization of a nested transaction. A traditional nested transaction is a transaction closure where the child transactions cannot be independent, i.e., a child transaction cannot leave the execution boundaries of its parent.

Definition 6.1 (Transaction Closures) *A transaction closure is a structured transaction where each (non-root) transaction is:*

- *vital, weak-vital, weak-non-vital, or non-vital, and*
- *dependent, weak-dependent, or independent, and*
- *inheriting or non-inheriting, and*
- *releasing or delegating, and*
- *sequential or parallel.*

Theoretically, all introduced kinds of transaction dependencies could be combined to form transactions of a transaction closure. Due to space restrictions, we cannot discuss all possible combinations. However, we can state that some combinations are incompatible or result in dependencies with other properties. To simplify the discussion, we only consider combined dependencies between a child t_j and its parent t_i. The presented results are also valid for superiors with the corresponding dependencies.

6.1 Abort Dependencies

The combination of upwards and downwards abort dependencies results in restricted termination event combinations and special execution orders in some cases.

Theorem 6.1 *Combining the vital dependency with the downwards abort dependencies results in an abort and commit dependent parent t_i and differ in the dependency of t_j in case t_i aborts*[4]*:*

[4]Due to space restrictions, we do not present the proofs of the theorems discussed in this section. However, the theorems immediately follow from the combinations of the definitions given in the previous sections.

parent t_i	a_{t_i}	a_{t_i}	c_{t_i}	c_{t_i}
child t_j	a_{t_j}	c_{t_j}	a_{t_j}	c_{t_j}
$\mathsf{vital}(t_i,t_j) \wedge \mathsf{dep}(t_i,t_j)$	true	false	false	$\mathsf{c}_{t_j} \to \mathsf{c}_{t_i}$
$\mathsf{vital}(t_i,t_j) \wedge \mathsf{weak\text{-}dep}(t_i,t_j)$	true	$\mathsf{c}_{t_j} \to \mathsf{a}_{t_i}$	false	$\mathsf{c}_{t_j} \to \mathsf{c}_{t_i}$
$\mathsf{vital}(t_i,t_j) \wedge \mathsf{indep}(t_i,t_j)$	true	true	false	$\mathsf{c}_{t_j} \to \mathsf{c}_{t_i}$
$\mathsf{weak\text{-}vital}(t_i,t_j) \wedge \mathsf{dep}(t_i,t_j)$	true	false	$\mathsf{c}_{t_k} \to \mathsf{c}_{t_i}$	$\mathsf{c}_{t_j} \to \mathsf{c}_{t_i}$
$\mathsf{weak\text{-}vital}(t_i,t_j) \wedge \mathsf{weak\text{-}dep}(t_i,t_j)$	true	$\mathsf{c}_{t_j} \to \mathsf{a}_{t_i}$	$\mathsf{c}_{t_k} \to \mathsf{c}_{t_i}$	$\mathsf{c}_{t_j} \to \mathsf{c}_{t_i}$
$\mathsf{weak\text{-}vital}(t_i,t_j) \wedge \mathsf{indep}(t_i,t_j)$	true	true	$\mathsf{c}_{t_k} \to \mathsf{c}_{t_i}$	$\mathsf{c}_{t_j} \to \mathsf{c}_{t_i}$
$\mathsf{weak\text{-}non\text{-}vital}(t_i,t_j) \wedge \mathsf{dep}(t_i,t_j)$	true	false	$\mathsf{a}_{t_j} \to \mathsf{c}_{t_i}$	$\mathsf{c}_{t_j} \to \mathsf{c}_{t_i}$
$\mathsf{weak\text{-}non\text{-}vital}(t_i,t_j) \wedge \mathsf{weak\text{-}dep}(t_i,t_j)$	true	$\mathsf{c}_{t_j} \to \mathsf{a}_{t_i}$	$\mathsf{a}_{t_j} \to \mathsf{c}_{t_i}$	$\mathsf{c}_{t_j} \to \mathsf{c}_{t_i}$
$\mathsf{weak\text{-}non\text{-}vital}(t_i,t_j) \wedge \mathsf{indep}(t_i,t_j)$	true	true	$\mathsf{a}_{t_j} \to \mathsf{c}_{t_i}$	$\mathsf{c}_{t_j} \to \mathsf{c}_{t_i}$
$\mathsf{non\text{-}vital}(t_i,t_j) \wedge \mathsf{dep}(t_i,t_j)$	true	false	true	true
$\mathsf{non\text{-}vital}(t_i,t_j) \wedge \mathsf{weak\text{-}dep}(t_i,t_j)$	true	$\mathsf{c}_{t_j} \to \mathsf{a}_{t_i}$	true	true
$\mathsf{non\text{-}vital}(t_i,t_j) \wedge \mathsf{indep}(t_i,t_j)$	true	true	true	true
$\mathsf{del}(t_i,t_j) \wedge \mathsf{non\text{-}vital}(t_i,t_j) \wedge \mathsf{dep}(t_i,t_j)$	true	false	$\mathsf{a}_{t_j} \to \mathsf{c}_{t_i}$	$\mathsf{c}_{t_j} \to \mathsf{c}_{t_i}$
$\mathsf{del}(t_i,t_j) \wedge \mathsf{non\text{-}vital}(t_i,t_j) \wedge \mathsf{weak\text{-}dep}(t_i,t_j)$	true	$\mathsf{c}_{t_j} \to \mathsf{a}_{t_i}$	$\mathsf{a}_{t_j} \to \mathsf{c}_{t_i}$	$\mathsf{c}_{t_j} \to \mathsf{c}_{t_i}$
$\mathsf{del}(t_i,t_j) \wedge \mathsf{non\text{-}vital}(t_i,t_j) \wedge \mathsf{indep}(t_i,t_j)$	true	$\mathsf{c}_{t_j} \to \mathsf{a}_{t_i}$	$\mathsf{a}_{t_j} \to \mathsf{c}_{t_i}$	$\mathsf{c}_{t_j} \to \mathsf{c}_{t_i}$
$\mathsf{seq}(t_i,t_j) \wedge \mathsf{vital}(t_i,t_j) \wedge \mathsf{dep}(t_i,t_j)$	true	false	false	false
$\mathsf{seq}(t_i,t_j) \wedge \mathsf{vital}(t_i,t_j) \wedge \mathsf{weak\text{-}dep}(t_i,t_j)$	true	false	false	false
$\mathsf{seq}(t_i,t_j) \wedge \mathsf{vital}(t_i,t_j) \wedge \mathsf{indep}(t_i,t_j)$	true	true	false	false

Table 6 Combined Dependencies

$$\begin{aligned}
\mathsf{vital}(t_i,t_j) \wedge \mathsf{dep}(t_i,t_j) &\Rightarrow (t_i\,\mathcal{AD}\,t_j) \wedge (t_i\,\mathcal{CD}\,t_j) \\
&\quad \wedge (t_j\,\mathcal{AD}\,t_i) \\
\mathsf{vital}(t_i,t_j) \wedge \mathsf{weak\text{-}dep}(t_i,t_j) &\Rightarrow (t_i\,\mathcal{AD}\,t_j) \wedge (t_i\,\mathcal{CD}\,t_j) \\
&\quad \wedge (t_j\,\mathcal{WD}\,t_i) \\
\mathsf{vital}(t_i,t_j) \wedge \mathsf{indep}(t_i,t_j) &\Rightarrow (t_i\,\mathcal{AD}\,t_j) \wedge (t_i\,\mathcal{CD}\,t_j)
\end{aligned}$$

The termination events of the transactions t_i and t_j are represented in Table 6[5]. In the case of a vital child t_j, t_i is not allowed to commit while t_j abort. The commit of the child has to precede the commit of the parent. The downwards abort dependencies influence the case in which the child aborts and the parent commits (see the third column of Table 6).

Theorem 6.2 *Combining the weak-vital dependency with the downwards abort dependencies:*

$$\begin{aligned}
&\mathsf{weak\text{-}vital}(t_i,t_j) \wedge \mathsf{dep}(t_i,t_j) \Rightarrow (t_i\,\mathcal{CD}\,t_j) \wedge \\
&\quad (\exists t_k : \mathsf{cont}(t_k,t_j) \wedge \mathsf{a}_{t_j} \Rightarrow (t_i\,\mathcal{BBCD}\,t_k)) \wedge (t_j\,\mathcal{AD}\,t_i) \\
&\mathsf{weak\text{-}vital}(t_i,t_j) \wedge \mathsf{weak\text{-}dep}(t_i,t_j) \Rightarrow (t_i\,\mathcal{CD}\,t_j) \wedge \\
&\quad (\exists t_k : \mathsf{cont}(t_k,t_j) \wedge \mathsf{a}_{t_j} \Rightarrow (t_i\,\mathcal{BBCD}\,t_k)) \wedge (t_j\,\mathcal{WD}\,t_i) \\
&\mathsf{weak\text{-}vital}(t_i,t_j) \wedge \mathsf{indep}(t_i,t_j) \Rightarrow (t_i\,\mathcal{CD}\,t_j) \wedge \\
&\quad (\exists t_k : \mathsf{cont}(t_k,t_j) \wedge \mathsf{a}_{t_j} \Rightarrow (t_i\,\mathcal{BBCD}\,t_k))
\end{aligned}$$

The weak-vital property forces that the parent is commit dependent on the child transaction and, moreover, that there exists one alternative transaction which is executed in case the child aborts and which must be committed in order the parent can commit.

The weak-vital dependency allows that the parent t_i commits and t_j aborts (see Table 6). However, in this case an alternative transaction t_k has to be initiated and committed before the parent can commit.

[5]Termination event combinations which are valid in both execution orders are denoted as true and invalid termination event combinations are denoted as false. There are termination event combinations which are only valid if they are executed in a special order. In this case the corresponding execution order is noted.

Theorem 6.3 *The combination of the weak-non-vital dependency with the downwards abort dependencies:*

$$\begin{aligned}
\mathsf{weak\text{-}non\text{-}vital}(t_i,t_j) \wedge \mathsf{dep}(t_i,t_j) &\Rightarrow (t_i\,\mathcal{CTD}\,t_j) \\
&\quad \wedge (t_j\,\mathcal{AD}\,t_i) \\
\mathsf{weak\text{-}non\text{-}vital}(t_i,t_j) \wedge \mathsf{weak\text{-}dep}(t_i,t_j) &\Rightarrow (t_i\,\mathcal{CTD}\,t_j) \\
&\quad \wedge (t_j\,\mathcal{WD}\,t_i) \\
\mathsf{weak\text{-}non\text{-}vital}(t_i,t_j) \wedge \mathsf{indep}(t_i,t_j) &\Rightarrow (t_i\,\mathcal{CTD}\,t_j)
\end{aligned}$$

In contrast to a weak-vital or vital child, a weak-non-vital child requires that the abortion of the child precedes the commit of its parent (if this case occurs).

Finally, we consider the case that transaction t_j is non-vital for t_i. If t_j is further independent of t_i, there exists no restrictions on the termination order.

Theorem 6.4 *Combining the non-vital dependency with the downwards abort dependencies:*

$$\begin{aligned}
\mathsf{non\text{-}vital}(t_i,t_j) \wedge \mathsf{dep}(t_i,t_j) &\Rightarrow (t_j\,\mathcal{AD}\,t_i) \\
\mathsf{non\text{-}vital}(t_i,t_j) \wedge \mathsf{weak\text{-}dep}(t_i,t_j) &\Rightarrow (t_j\,\mathcal{WD}\,t_i) \\
\mathsf{non\text{-}vital}(t_i,t_j) \wedge \mathsf{indep}(t_i,t_j) &\Rightarrow \mathsf{true}
\end{aligned}$$

6.2 Object Visibility Dependencies

Abort dependencies are constraints on the occurrence of the termination events of related transactions. In contrast, object visibility dependencies have an influence on the view of the related transactions:

Combining delegating and inheriting: The view of a child t_j includes its access set and the view of the parent t_i which consists of the commit projection of the current history ($\mathsf{H}^c_{ct} = \{p_t[ob] \mid \mathit{committed}(t)\}$) and the access set of t_i ($\mathit{view}_{t_i} = \mathsf{H}^c_{ct} \cup \mathsf{AS}_{t_i}$). Thus, t_j can see the results of t_i and delegates its own access set with the commit to t_i. In consequence, the access set and, thus, the view of the parent t_i is extended by the access set of the committed child t_j. However, these results are not visible to (other) concurrent transactions.

Combining releasing and inheriting: The child t_j inherits the view of its parent t_i. Due to the releasing property, the results of the child t_j are immediately reflected in the commit projection of the current history with the commit of t_j. Thus, the view of the parent t_i is the same as in the case of a delegating child. However, the access set of t_i is not extended by the results of t_j.
Combining delegating and non-inheriting: A non-inheriting dependency reduces the view of a child t_j to the commit projection of the current history and its own access set. In other words, a child does not see the effects of its parent. A combination with the delegating property leads to an extension of the view of the parent t_i by delegating t_j's access set with its commit.
Combining non-inheriting and releasing: This combined dependency is equivalent to a dependency between two concurrent transactions of different transaction closures. Both transactions see only their own access set in addition to the commit projection of the current history, i.e., they do not see the effects of each other since the effects are committed.

	$view_{t_j}$ (at b_{t_j})	$view_{t_i}$ (at c_{t_j})
$\mathsf{inher}(t_i,t_j) \wedge \mathsf{del}(t_i,t_j)$	$\mathsf{H}^c_{ct} \cup \mathsf{AS}_{t_i}$	$\mathsf{H}^c_{ct} \cup \mathsf{AS}_{t_i} \cup \mathsf{AS}_{t_j}$
$\mathsf{inher}(t_i,t_j) \wedge \mathsf{rel}(t_i,t_j)$	$\mathsf{H}^c_{ct} \cup \mathsf{AS}_{t_i}$	$\mathsf{H}^c_{ct} \cup \mathsf{AS}_{t_i}$
$\mathsf{non\text{-}inher}(t_i,t_j) \wedge \mathsf{del}(t_i,t_j)$	H^c_{ct}	$\mathsf{H}^c_{ct} \cup \mathsf{AS}_{t_i} \cup \mathsf{AS}_{t_j}$
$\mathsf{non\text{-}inher}(t_i,t_j) \wedge \mathsf{rel}(t_i,t_j)$	H^c_{ct}	$\mathsf{H}^c_{ct} \cup \mathsf{AS}_{t_i}$

Table 7 Combined Object Visibility Dependencies

In Table 7 we compare the view of the child t_j at the starting time of t_j and the view of the parent t_i at the commit time of t_j for all combinations discussed above. Due to the consideration of the views of the related transactions at different points in time, the commit projection of the current history H^c_{ct} may be changed, e.g. with the commit of a releasing transaction.

6.3 Object Visibility and Abort Dependencies

The object visibility dependencies have also an influence on the abort dependencies. Concerning the delegating property, the delegation of the access set of t_j to its parent t_i has to precedes the termination of t_i. A combination with the independent property restricts the independent property to a weak-dependent property because the requirement that the commit of t_j must precede the abortion of t_i is added by the delegating dependency.

A combination of a non-vital dependency with the delegating property, converts the former one to a weak-non-vital dependency. This is due to fact that the commit of transaction t_j has to precede the commit of its parent t_i. The abortion of t_j has also to precede the commit of t_i because the parent has to wait for the termination decision of the child to commit.

Theorem 6.5 *Combining independent and non-vital with the delegating dependency leads to a transformation ($\rightsquigarrow$) to a weak-dependent and weak-non-vital dependency, respectively:*

$$\mathsf{indep}(t_i,t_j) \wedge \mathsf{del}(t_i,t_j) \rightsquigarrow \mathsf{weak\text{-}dep}(t_i,t_j)$$
$$\mathsf{non\text{-}vital}(t_i,t_j) \wedge \mathsf{del}(t_i,t_j) \rightsquigarrow \mathsf{weak\text{-}non\text{-}vital}(t_i,t_j)$$

Because of space restrictions, we only consider the combination of the delegating dependency with non-vital and the downwards abort dependencies (Table 6). We can easily see that the delegating dependency constrain all valid termination event combinations by adding that the child has to terminate before the parent.

In contrast, a releasing dependency requires no restriction on the termination order of the related transactions. Therefore, in this case, there are no additional constraints on the abort dependencies. The inheriting and non-inheriting properties are related to the begin of a child. Thus, these dependencies have also no influence on the abort dependencies of the related transactions.

6.4 Execution Order Dependencies

Execution order dependencies can be combined with the abort and object visibility dependencies. In general, parent and child transactions are executed in parallel. However, in advanced applications, e.g. in active databases, a transaction may be triggered within another transaction and executed sequentially after the termination of the triggering transaction.

Assume that a child t_j has to be executed sequentially after the termination of the parent t_i ($\mathsf{seq}(t_i,t_j)$):

(1) In case t_j has to be started after the commit of t_i ($\mathsf{seq\text{-}c}(t_i,t_j)$), the dependency between these two transaction should be non-vital. In case of the vital dependency, the abortion of the child may force the parent to abort, too. However, in combination with a sequential dependency, this is not reasonable because the parent terminates before the child begins executing. As shown in Table 6, the vital, weak-vital, and weak-non-vital dependency requires that the child t_j commits before the parent t_i can commit. However, this contradicts the requirement of the sequential dependency that t_j has to start after the termination of t_i. Thus, this combinations are incompatible.

(2) The same argument holds also for the case that the child t_j has to be started after the abortion of the parent t_i ($\mathsf{seq\text{-}a}(t_i,t_j)$). Here, the downwards abort dependency should be independent. According to Table 6, a weak-dependent transaction t_j requires that its commit precedes the abortion of its parent t_i. However, this also contradicts the requirement of the sequential dependency that t_j has to start after the termination of t_i. In case of dependent transactions, the case that t_i aborts and t_j commits is disallowed. Thus, the execution of t_j after the abortion of t_i cannot lead to a commit of t_j. However, the abortion of t_j is valid. Thus, the execution of transaction t_j makes no sense because it is aborted in any case.

Table 6 represents the combination of the sequential dependency with vital and the downwards abort dependencies. Analogously to the weak-vital and weak-non-vital dependency, a combination of vital and sequential leads to contradictions in the execution ordering requirements. Thus, the last column is completely adapted to false where in the second column only the combination with weak-dependent is changed to false.

The delegating dependency requires the delegation of the results of t_j to the parent t_i. From this follows, a sequential execution of t_j after the termination of t_i is impossible (analogously to the abort dependencies).

Finally, we can state that a sequential dependency between two transactions requires ($\curvearrowright$) that these transactions are also non-vital, independent, and releasing:

$$\mathsf{seq}(t_i, t_j) \curvearrowright \mathsf{non\text{-}vital}(t_i, t_j) \wedge \mathsf{indep}(t_i, t_j) \wedge \mathsf{rel}(t_i, t_j)$$

The combination of a sequential dependency with the other abort and object visibility dependencies are either not reasonable or (logically) impossible.

7 Exemplary Application of the Framework

Using the notion of transaction closures we are now able to specify advanced transaction models as well as detached transactions. In the literature (for a survey see (Elm92)), several advanced transaction models are proposed which base on the nested transaction model (Mos85). For instance, closed nested transactions are transaction closures which consist of transactions which are *vital*, *dependent*, *inheriting*, and *delegating* from the viewpoint of their parents. In the following, we concentrate on detached transactions.

The notion of *detached* transactions comes from the area of active databases where triggered actions are executed as separate transactions out of the scope of the triggering transaction. We will focus on the four detached modes described in (Buc94).

In our framework, the triggering transaction is the parent t_i of a triggered child t_j with special dependencies. Here, we have to consider two dimensions of dependencies: the execution order dependency and the commit dependency.

Detached transactions are executed as separate transactions. From this follows that a detached transaction is a *non-inheriting* and *releasing* transaction. It has its own scope and does not see the effects of its parents and other superiors until they are committed. A further implication is that detached transactions are *non-vital*, i.e., the abortion of a detached transaction has no influence on the parent and the transaction can leave the execution boundaries of the parent. The other dependencies like the downwards abort dependency or the execution order dependency differ in the four cases of detached transactions:

(1) A *detached independent* transaction is executed in parallel to the parent and is independent of the termination of the parent:

$\mathsf{non\text{-}inher}(t_i, t_j) \wedge \mathsf{rel}(t_i, t_j) \wedge \mathsf{non\text{-}vital}(t_i, t_j) \wedge$
$\mathsf{par}(t_i, t_j) \wedge \mathsf{indep}(t_i, t_j)$

(2) A *detached parallel causally dependent* transaction is executed parallel to the parent transaction, but it may only commit if the parent commits. Thus, a detached parallel causally dependent transaction is abort dependent on the parent:

$\mathsf{non\text{-}inher}(t_i, t_j) \wedge \mathsf{rel}(t_i, t_j) \wedge \mathsf{non\text{-}vital}(t_i, t_j) \wedge$
$\mathsf{par}(t_i, t_j) \wedge \mathsf{dep}(t_i, t_j)$

(3) A *detached sequential causally dependent* transaction is only executed if the parent transaction commits. Hence, there exists no abort dependency between a detached transaction and its parent:

$\mathsf{non\text{-}inher}(t_i, t_j) \wedge \mathsf{rel}(t_i, t_j) \wedge \mathsf{non\text{-}vital}(t_i, t_j) \wedge$
$\mathsf{seq\text{-}c}(t_i, t_j) \wedge \mathsf{indep}(t_i, t_j)$

(4) A *detached exclusive but causally dependent* transaction which is executed sequential after an abortion of the parent has to be independent of the parent's abortion. This type is called exclusive because the detached transaction may only commit if the parent aborts:

$\mathsf{non\text{-}inher}(t_i, t_j) \wedge \mathsf{rel}(t_i, t_j) \wedge \mathsf{non\text{-}vital}(t_i, t_j) \wedge$
$\mathsf{seq\text{-}a}(t_i, t_j) \wedge \mathsf{indep}(t_i, t_j)$

In conclusion, we have shown that the proposed extended framework allows an easy and elegant formalization of non-traditional transaction models which go far beyond the nested transaction model.

8 Conclusions and Outlook

In this paper, we presented a generalized framework for describing and classifying dependent transactions. The concept of *transaction closures* extends the concept of nested transaction for example allowing detached child transactions in such transaction closures. For formally describing transaction closures, we extended the ACTA framework. The resulting framework gives a richer repertoire for classifying dependency relations between transactions inside a transaction closure. These extensions may become relevant for example in the application areas of active databases and database federations, where classical nested transactions are too restrictive because a child of a nested transaction tree has to terminate in the scope of its parent transaction.

Our future work will concentrate on dependency enforcement in transaction closures. We will attempt to adopt the methods proposed in (GHK93) to our framework and provide some extensions to capture the transitive properties of the transaction dependencies.

REFERENCES

A. Buchmann, M. T. Özsu, M. Hornick, D. Georgakopoulos, and F. Manola. A Transaction Model for Active Distributed Object Systems. In (Elm92), pp. 123–151.

A. P. Buchmann. Active Object Systems. In A. Dogac, M. T. Özsu, A. Biliris, and T. Sellis (eds.), *Advances in Object-Oriented Database Systems*, Nato ASI Series, pp. 201–224. Springer, 1994.

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

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 VLDB*, pp. 113–122. Morgan Kaufmann, 1991.

U. Dayal, E. Hanson, and J. Widom. Active Database Systems. In W. Kim (ed.), *Modern Database Systems*, pp. 434–456. ACM Press, 1995.

A. K. Elmagarmid (ed.). *Database Transaction Models For Advanced Applications*. Morgan Kaufmann, 1992.

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*, pp. 253–257, IEEE Computer Society Press, 1993.

J. E. B. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing.* MIT Press, 1985.

M. Rusinkiewicz and A. Sheth. Specification and Execution of Transactional Workflows. In W. Kim (ed.), *Modern Database Systems*, pp. 593–620, ACM Press, 1995.

H. Wächter and A. Reuter. The ConTract Model. In (Elm92), pp. 219–263.

G. Weikum and H.-J. Schek. Concepts and Applications of Multi-level Transactions and Open Nested Transactions. In (Elm92), pp. 515–553.