To appear in: W. Litwin, T. Morzy and G. Vossen, editors, Second East-European Symposium on Advances in Databases and Information Systems (ADBIS'98), September 8-11, 1998, Poznan, Poland, LNCS, Springer-Verlag, Berlin, 1998.

Extending Transaction Closures by N-ary Termination Dependencies

Kerstin Schwarz, Can Türker, and Gunter Saake

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

Abstract. Transaction dependencies have been recognized as a valuable method in describing restrictions on the executions of sets of transactions. A transaction closure is a generalized transaction structure consisting of a set of related transactions which are connected by special dependencies. Traditionally, relationships between transactions are formulated by *binary* dependencies. However, there are applications scenarios where dependencies must be specified among more than two transactions. Since *n-ary* dependencies cannot be expressed by binary dependencies, appropriate extensions are required. In this paper, we extend the concept of transaction closure by *ternary* termination dependencies. We show how n-ary termination dependencies can be expressed by binary and ternary termination dependencies. As a result, we present rules for reasoning about the combination of these termination dependencies.

1 Introduction

The concept of transaction closure [10] provides a uniform framework for describing advanced transactions and activities. A transaction closure consists of a set of related transactions which are connected by dependencies. By using special dependencies we are able to model classical nested transactions as well as activities where a transaction may survive the termination of its parent transaction. This is needed for example in models for long-during activities [4, 9], workflows [6], or transactions in active databases [7, 2]. The concept of transaction closure enables us to model such advanced applications in a modular way.

Transaction dependencies have been recognized as a valuable method in describing certain restrictions on the executions of sets of transactions [1, 12]. In [11] we have introduced different kinds of binary dependencies. We distinguished between termination, object visibility, and execution dependencies. However, there are application scenarios which cannot be modeled by only binary dependencies. This is due to the fact that in general *n-ary* dependencies which describe relationships among more than two transactions cannot be expressed by a set

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

of binary dependencies. For instance, the fact that a transaction t_i has to be aborted in case two other transactions t_j and t_k abort can only be expressed by a ternary dependency.

In this paper, we concentrate on termination dependencies and extend our framework by n-ary termination dependencies. First, we investigate n-ary relationships among transactions and define a set of *ternary* dependencies. We show that each kind of n-ary dependency can be expressed by a combination of binary and ternary dependencies. Since ternary dependencies may be influenced by other ternary or binary dependencies defined on the same (sub)set of transactions, we have to analyze the connection between binary and ternary dependencies. Some of these dependency combinations are valid and reasonable; others are incompatible in the sense that the ternary dependency is relaxed by the binary dependencies. As a result, we derive rules for reasoning about the combination of two dependencies. Using these rules we are able to state how two or more transactions are interrelated.

The paper is organized as follows: Sect. 2 provides the basic notions whereas Sect. 3 introduces ternary termination dependencies. In Sect. 4, we show how n-ary termination dependencies can be expressed by binary and ternary dependencies. In Sect. 5, the ternary dependencies among three transactions are combined with binary termination dependencies between pairs of the corresponding transactions. An example scenario of a transaction closure containing binary and ternary dependencies is considered in Sect. 6.

2 Foundations

Traditionally, a *transaction* is an execution unit consisting of a set of database operations. A transaction t_i is started by invoking the primitive *begin* (b_{t_i}) and is terminated by either *commit* (c_{t_i}) or *abort* (a_{t_i}). These primitives are termed as *significant events* [3]. A transaction invokes operations, termed as *object events*, to access and manipulate the state of database objects. A *history* of a concurrent execution of a set of transactions T comprises all events associated with the transactions in T and indicates the (partial) order in which these events occur. The complete history contains only terminated transactions and is denoted as H, the current (incomplete) history is termed as H_{ct}.

A set of transactions with dependencies among them can be considered as *transaction closure* [10, 11]. The concept of transaction closure is a generalization of the well-known concept of nested transactions [8]. Each transaction closure consist of a set of transactions. Exactly one of these transactions is denoted as root transaction. A root transaction is a transaction which has no parent. Each non-root transaction has exactly one parent and its initiation must follow the initiation of the parent. Each transaction closure is acyclic. The effects of transactions on other transactions are described by dependencies which are constraints on possible histories. We distinguish between *termination*, *object visibility*, and *execution dependencies*.

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}).

These termination event combinations may be valid in any order (denoted by ✓) or are not valid (denoted by —). As depicted in Table 1, we identify five dependencies as reasonable according to real-world application semantics. The termination dependency between t_i and t_j is called *vital-dependent*, denoted as $vital\_dep(t_i, t_j)$, if the transactions are *abort-dependent* on each other. That is, the abort of transaction t_i leads to the abort of t_j and vice versa. Thus, either the transactions commit together or both abort. The vital-dependent dependency is (as the name suggests) a combination of the dependencies *vital* and *dependent*. The *vital* dependency between two transactions t_i and t_j, denoted as $vital(t_i, t_j)$, concerns the case where the abort of transaction t_i leads to the abort of t_j. In contrast, a *dependent* transaction t_j has to abort if t_i aborts. This fact is defined as $dep(t_i, t_j)$. Two transactions are called *exclusive* dependent on each other, denoted as $exc(t_i, t_j)$, if only one of the transactions is allowed to finish successfully. Our fifth dependency concerns the case where each combination of transaction termination events is valid. Therefore, the involved transactions t_i and t_j are called *independent*, denoted as $indep(t_i, t_j)$. For a formal definition of the termination dependencies and more detail information see [11].

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}	✓	✓	✓	✓	✓
a_{t_i}	c_{t_j}	—	—	✓	✓	✓
c_{t_i}	a_{t_j}	—	✓	—	✓	✓
c_{t_i}	c_{t_j}	✓	✓	✓	—	✓

Table 1. Termination Dependencies between two Transactions t_i and t_j

3 Ternary Termination Dependencies

Relations between activities refer to dependencies between transactions of a transaction closures. In [11] we defined different binary transaction termination dependencies such as *vital, dependent, vital-dependent, exclusive,* and *independent*. Obviously, we cannot express all possible kinds of relations among transactions using only binary dependencies as illustrated by the following example.

Example 1. Assume, we want to book a room in one of the hotels Hilton and Maritim, respectively, and hire a car. Booking a room in the Hilton hotel is realized by transaction t_i and in the Maritim hotel by transaction t_j. Hiring a car is represented by transaction t_k. In case we cannot book a room in one of the hotels (both t_i and t_j abort), we do not need to hire a car (abort of t_k).

An attempt to model this situation by binary dependencies may be first to cancel the hiring of the car in case a room in the Hilton hotel is not available and second to cancel the hiring of the car if we cannot book a room in the Maritim hotel. This is equivalent to the dependencies $vital(t_i, t_k)$ and $vital(t_j, t_k)$. However, this specification would lead to an abortion of the car hiring (transaction t_k) in case either t_i or t_j aborts (which means that one room cannot be booked). Consequently, a car can only be rent in case a room in both hotels is booked. This contradicts the intended semantics of the example scenario. □

Thus, statements such as *"if both transactions t_i and t_j aborts, then transaction t_k has to abort, too"* corresponds to the following formulas[1] which cannot be expressed by binary dependencies:

$$((a_{t_i} \wedge a_{t_j}) \Rightarrow a_{t_k}) \equiv (a_{t_i} \Rightarrow (a_{t_j} \Rightarrow a_{t_k})) \tag{1}$$

Therefore, we have to extend our framework by ternary termination dependencies. Formula (1) forces the abort of transaction t_k in case both transactions t_i and t_j abort. However, the abort of t_k may also be the consequence of the commit of one the transactions t_i and t_j. We define the following ternary dependencies:

Definition 1 (Vital). *The predicate $vital(t_i, t_j, t_k)$ is true if and only if transaction t_k has to abort in case both transactions t_i and t_j abort:*

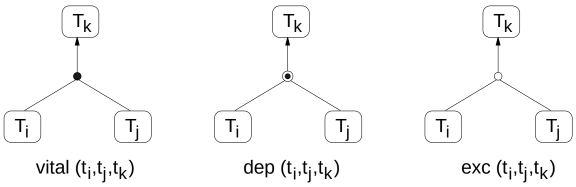
$$vital(t_i, t_j, t_k) :\Leftrightarrow (a_{t_i} \wedge a_{t_j}) \Rightarrow a_{t_k}$$

Definition 2 (Dependent). *The predicate $dep(t_i, t_j, t_k)$ is true if and only if transaction t_k has to abort in case transaction t_i commits and transaction t_j aborts:*

$$dep(t_i, t_j, t_k) :\Leftrightarrow (c_{t_i} \wedge a_{t_j}) \Rightarrow a_{t_k}$$

Definition 3 (Exclusive). *The predicate $exc(t_i, t_j, t_k)$ is true if and only if transaction t_k has to abort in case both transactions t_i and t_j commit:*

$$exc(t_i, t_j, t_k) :\Leftrightarrow (c_{t_i} \wedge c_{t_j}) \Rightarrow a_{t_k}$$



[1] For improving the readability we simplify the predicate $(a_{t_i} \in H)$ to (a_{t_i}). This is also done for the other transactions and termination events.

Table 2 illustrates the termination events of the transactions t_i, t_j, and t_k. The dependencies introduced so far disallow exactly one termination event combination. Ternary vital dependencies disallow the cases where two transactions abort and one commits, e.g. $vital(t_i, t_j, t_k)$ disallows that t_i and t_j abort whereas t_k commits. Ternary dependent transactions disallow a case where two transactions commit and one aborts. In contrast, ternary exclusive dependencies disallow the case where all related transactions commit.

Theorem 1. *The following relationships immediately follows from the Definitions 1 to 3 (Therefore, we omit the (trivial) proofs):*

$$vital(t_i, t_j, t_k) \equiv vital(t_j, t_i, t_k) \tag{2}$$

$$dep(t_i, t_j, t_k) \equiv dep(t_k, t_j, t_i) \tag{3}$$

$$exc(t_i, t_j, t_k) \equiv exc(t_i, t_k, t_j) \ \equiv exc(t_k, t_i, t_j) \tag{4}$$

A binary dependency between t_i and t_j either disallows no $(indep(t_i, t_j))$, two or four $(vital\_dep(t_i, t_j))$ termination event combinations among three transactions t_i, t_j, and t_k. In contrast, an arbitrary termination event combination can be disallowed using the introduced ternary dependencies. Thus, this set of ternary dependencies is minimal. All other kinds of dependencies among three transactions can be expressed by combinations of ternary and binary dependencies.

	a_{t_i} a_{t_j} a_{t_k}	a_{t_i} a_{t_j} c_{t_k}	a_{t_i} c_{t_j} a_{t_k}	a_{t_i} c_{t_j} c_{t_k}	c_{t_i} a_{t_j} a_{t_k}	c_{t_i} a_{t_j} c_{t_k}	c_{t_i} c_{t_j} a_{t_k}	c_{t_i} c_{t_j} c_{t_k}
$vital(t_i, t_j, t_k)$	✓	—	✓	✓	✓	✓	✓	✓
$vital(t_i, t_k, t_j)$	✓	✓	—	✓	✓	✓	✓	✓
$vital(t_k, t_j, t_i)$	✓	✓	✓	✓	—	✓	✓	✓
$dep(t_i, t_j, t_k)$	✓	✓	✓	✓	✓	—	✓	✓
$dep(t_j, t_i, t_k)$	✓	✓	✓	—	✓	✓	✓	✓
$dep(t_i, t_k, t_j)$	✓	✓	✓	✓	✓	✓	—	✓
$exc(t_i, t_j, t_k)$	✓	✓	✓	✓	✓	✓	✓	—

Table 2. Termination Event Combinations of Ternary Termination Dependencies

4 N-ary Termination Dependencies

Additionally to binary and ternary dependencies, there may be relationships among more than three transactions. Such n-ary dependencies can be expressed by binary and ternary dependencies. This is done by decomposing an n-ary dependency into a set of binary and ternary dependencies which are connected by so-called dummy-transactions. For example, the 5-ary dependency

$$vital(t_i, t_j, t_k, t_l, t_m) \Leftrightarrow ((a_{t_i} \wedge a_{t_j} \wedge a_{t_k} \wedge a_{t_l}) \Rightarrow a_{t_m})$$

can be expressed by the following ternary dependencies including the dummy transactions t_a and t_b:

$$(a_{t_i} \wedge a_{t_j}) \Leftrightarrow a_{t_a}$$
$$(a_{t_k} \wedge a_{t_l}) \Leftrightarrow a_{t_b}$$
$$(a_{t_a} \wedge a_{t_b}) \Rightarrow a_{t_m}$$

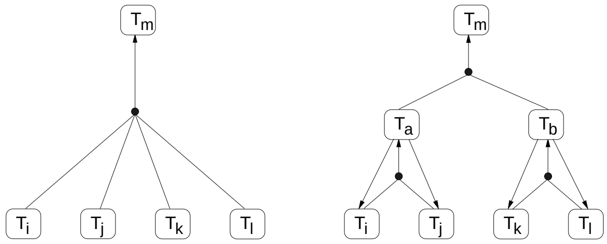
An equivalence expression can be transformed into a conjunction of binary and ternary dependencies as follows:

$$\begin{aligned}((a_{t_i} \wedge a_{t_j}) \Leftrightarrow a_{t_a}) &\equiv ((a_{t_i} \wedge a_{t_j}) \Rightarrow a_{t_a}) \wedge (a_{t_a} \Rightarrow (a_{t_i} \wedge a_{t_j}))) \\ &\equiv \underbrace{((a_{t_i} \wedge a_{t_j}) \Rightarrow a_{t_a})}_{vital(t_i,t_j,t_a)} \wedge \underbrace{(a_{t_a} \Rightarrow a_{t_i})}_{vital(t_a,t_i)} \wedge \underbrace{(a_{t_a} \Rightarrow a_{t_j})}_{vital(t_a,t_j)})\end{aligned}$$

Analogously, the second equivalence is transformed into a set of binary and ternary dependencies among the transactions t_k, t_l, and t_b:

$$\begin{aligned}((a_{t_k} \wedge a_{t_l}) \Leftrightarrow a_{t_b}) &\equiv ((a_{t_k} \wedge a_{t_l}) \Rightarrow a_{t_b}) \wedge (a_{t_b} \Rightarrow (a_{t_k} \wedge a_{t_l}))) \\ &\equiv \underbrace{((a_{t_k} \wedge a_{t_l}) \Rightarrow a_{t_b})}_{vital(t_k,t_l,t_b)} \wedge \underbrace{(a_{t_b} \Rightarrow a_{t_k})}_{vital(t_b,t_k)} \wedge \underbrace{(a_{t_b} \Rightarrow a_{t_l})}_{vital(t_b,t_l)})\end{aligned}$$

In summary, the 5-ary dependency and its equivalent representation with only ternary and binary dependencies is illustrated below. A binary dependency like $vital(t_a, t_i)$ is represented by an arrow from t_a to t_i:



The method of decomposition can also be used for ternary dependent and exclusive dependencies. Rules for reasoning about the transitive relationship between two arbitrary transactions of a transaction closure can be generated considering the introduced termination dependencies. In case of n-ary dependencies, we have to consider the combination and transitive relation of each type of n-ary dependency. However, this is too complex. N-ary dependencies are easier to handle when they are mapped onto binary and ternary dependencies.

The following example illustrates the decomposition of a 4-ary dependency which contains a disjunction of events on the right-hand side of the implication:

$$((a_{t_i} \wedge a_{t_j}) \Rightarrow (a_{t_m} \vee a_{t_n})) \equiv (c_{t_i} \vee c_{t_j} \vee a_{t_m} \vee a_{t_n}) \equiv \underbrace{((a_{t_i} \wedge a_{t_j} \wedge c_{t_m}) \Rightarrow a_{t_n})}_{vital(t_i,t_j,t_k,t_m)}$$

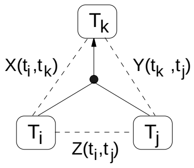
The resulting expression $((a_{t_i} \wedge a_{t_j} \wedge c_{t_m}) \Rightarrow a_{t_n})$ can be decomposed to a ternary dependency involving the dummy transaction t_a:

$$(a_{t_i} \wedge a_{t_j}) \Leftrightarrow a_{t_a}$$
$$(a_{t_a} \wedge c_{t_m}) \Rightarrow a_{t_n}$$

Using the method described above, each n-ary dependency can be decomposed into a set of binary and ternary dependencies with dummy transactions.

5 Combining Binary and Ternary Dependencies

In this section, we consider ternary termination dependencies among the transactions t_i, t_j, and t_k in combination with binary dependencies between each related transaction pair. We start with the ternary vital dependency. As depicted below, the ternary dependency among transactions t_i, t_j, and t_k is $vital(t_i, t_j, t_k)$, the transactions t_i and t_k are related over the dependency $X(t_i, t_k)$, t_k and t_j over dependency $Y(t_k, t_j)$, and t_i and t_j over dependency $Z(t_i, t_j)$:

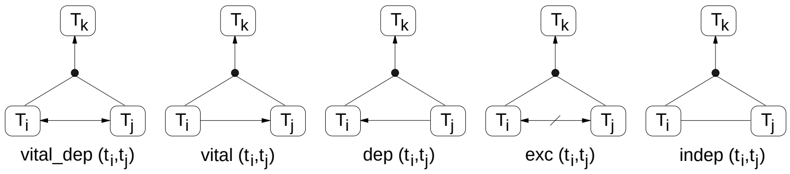


The binary termination dependencies are graphically represented as follows:

$$\begin{aligned} vital(t_i, t_j) &\text{ corresponds to } t_i \longrightarrow t_j \\ dep(t_i, t_j) &\text{ corresponds to } t_i \longleftarrow t_j \\ vital\_dep(t_i, t_j) &\text{ corresponds to } t_i \longleftrightarrow t_j \\ exc(t_i, t_j) &\text{ corresponds to } t_i \not\longleftrightarrow t_j \\ indep(t_i, t_j) &\text{ corresponds to } t_i \longrule t_j \end{aligned}$$

5.1 Considering Dependency $\mathbf{Z(t_i, t_j)}$

We start with the discussion of the dependency $Z(t_i, t_j)$ in combination with the ternary vital dependency $vital(t_i, t_j, t_k)$. We show that there are dependency combinations where the ternary dependency can be substituted by a binary dependency without changing the effect of the dependency combination. The dependency $Z(t_i, t_j)$ can be one of the five termination dependencies:



1. $\mathbf{vital\_dep(t_i, t_j) \wedge vital(t_i, t_j, t_k)}$: The ternary dependency $vital(t_i, t_j, t_k)$ states that t_k has to abort if both t_i and t_j abort. Due to $vital\_dep(t_i, t_j)$, both transactions t_i and t_j abort if either t_i or t_j aborts. For expressing such a relationship among three transactions we do not need a ternary dependency:

$$(\underbrace{(a_{t_i} \Rightarrow a_{t_j} \wedge a_{t_j} \Rightarrow a_{t_i})}_{vital\_dep(t_i,t_j)} \wedge \underbrace{((a_{t_i} \wedge a_{t_j}) \Rightarrow a_{t_k})}_{vital(t_i,t_j,t_k)}) \equiv (\underbrace{(a_{t_i} \Rightarrow a_{t_j} \wedge a_{t_j} \Rightarrow a_{t_i})}_{vital\_dep(t_i,t_j)} \wedge \underbrace{(a_{t_i} \Rightarrow a_{t_k})}_{vital(t_i,t_k)})$$

An alternative to force t_k's abort is to extend $vital\_dep(t_i, t_j)$ by $dep(t_k, t_j)$.

2. $\mathbf{vital(t_i, t_j) \wedge vital(t_i, t_j, t_k)}$: The ternary dependency $vital(t_i, t_j, t_k)$ requires the abortion of transaction t_k if both transactions t_i and t_j abort; $vital(t_i, t_j)$ forces transaction t_k to abort if t_i aborts. As in the case before, the same situation can be modelled without the ternary dependency, for instance, as follows:

$$(\underbrace{(a_{t_i} \Rightarrow a_{t_j})}_{vital(t_i,t_j)} \wedge \underbrace{((a_{t_i} \wedge a_{t_j}) \Rightarrow a_{t_k})}_{vital(t_i,t_j,t_k)}) \equiv (\underbrace{(a_{t_i} \Rightarrow a_{t_j})}_{vital(t_i,t_j)} \wedge \underbrace{(a_{t_i} \Rightarrow a_{t_k})}_{vital(t_i,t_k)})$$

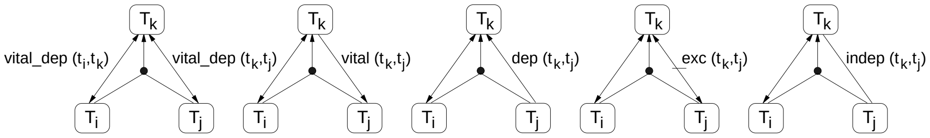
3. $\mathbf{dep(t_i, t_j) \wedge vital(t_i, t_j, t_k)}$: Since $vital(t_i, t_j) \equiv dep(t_j, t_i)$ holds [11], the result is similar to the previous case. The only difference is that here transaction t_j (instead of t_i) force the abortion of the other transactions.

4. $\mathbf{exc(t_i, t_j) \wedge vital(t_i, t_j, t_k)}$: Due to the $exc(t_i, t_j)$, one of the transactions t_i and t_j has to abort in case the other one commits. The ternary vital dependency cause the abortion of transaction t_k, if both t_i and t_j abort. In this case, we cannot express such as relationship by binary dependencies. For instance, the transactions t_i and t_j book a room in different hotels whereas transaction t_k prints the invoice. In case one room is booked, the other booking transaction has to be canceled. This is modeled by the exclusive dependency. On the other hand, if we cannot book a room in any hotel, then the whole transaction closure has to be canceled. This is specified by the ternary vital dependency.

5. $\mathbf{indep(t_i, t_j) \wedge vital(t_i, t_j, t_k)}$: Independent transactions have no influence on each other. A ternary vital dependency is an additional restriction on the execution of the corresponding transactions.

5.2 Considering the Dependencies $\mathbf{X(t_i, t_k)}$ and $\mathbf{Y(t_k, t_j)}$

The dependency $vital(t_i, t_j, t_k)$ disallows the case where transaction t_k commits and the transactions t_i and t_j abort. In the sequel, we consider how far the dependencies $X(t_i, t_k)$ and $Y(t_k, t_j)$ already avoid this case. So, we do not need to specify a "redundant" ternary vital dependency.



1. vital-dependent: As depicted above, the ternary vital dependency is already fulfilled if one of the dependencies $X(t_i, t_k)$ and $Y(t_k, t_j)$ is vital-dependent. That is, the termination event combination $(a_{t_i}, c_{t_k}, a_{t_j})$ is already disallowed. The abortion of t_i or t_j leads to the abortion of t_k in case of $vital\_dep(t_i, t_k)$ and $vital\_dep(t_k, t_j)$, respectively. In this case, transaction t_k cannot commit. Thus, the dependency $vital(t_i, t_j, t_k)$ does not put further constraints on the termination events of the related transactions.

2. vital/dependent: The termination event combination $(a_{t_i}, c_{t_k}, a_{t_j})$ is already disallowed by the dependencies $vital(t_i, t_k)$ and $dep(t_k, t_j)$. In contrast, the dependencies $vital(t_k, t_i)$ and $dep(t_j, t_k)$ specify that the transactions t_i and t_j, respectively, are aborted in case t_k aborts. In these cases, the ternary vital dependency is a further constraint on the execution of the related transactions.

3. exclusive/independent: Exclusive dependencies for $X(t_i, t_k)$ and $Y(t_k, t_j)$ cannot avoid the case in which transaction t_k commits and the other transactions abort. Therefore, in this case we need the ternary dependency to force t_k's abort in case of the abort of both t_i and t_j. The same is valid for independent transactions which do not influence each other.

5.3 Combining the Dependencies $\mathbf{X(t_i, t_k)}$, $\mathbf{Y(t_k, t_j)}$, and $\mathbf{Z(t_i, t_j)}$

Up to now, we investigated the dependencies $X(t_i, t_k)$, $Y(t_k, t_j)$, and $Z(t_i, t_j)$ in combination with the ternary vital dependency. As a result we obtained the following dependencies which can be (reasonable) combined with $vital(t_i, t_j, t_k)$:

$$X(t_i, t_k) : indep(t_i, t_k),\ exc(t_i, t_k),\ dep(t_i, t_k) \tag{5}$$

$$Y(t_k, t_j) : indep(t_k, t_j),\ exc(t_k, t_j),\ vital(t_k, t_j) \tag{6}$$

$$Z(t_i, t_j) : indep(t_i, t_j),\ exc(t_i, t_j) \tag{7}$$

Due to space restrictions, we summarize the valid dependency combinations for $dep(t_i, t_j, t_k)$ and $exc(t_i, t_j, t_k)$ and only discuss some interesting cases.

The dependency $dep(t_i, t_j, t_k)$ forces the abort of t_k in case t_i commits and t_j aborts. If dependency $Z(t_i, t_j)$ is $vital\_dep(t_i, t_j)$ or $dep(t_i, t_j)$, then the case that t_i commits and t_j aborts cannot occur. The dependencies $vital\_dep(t_i, t_k)$ and $dep(t_i, t_k)$ require t_i's abort in case t_k abort. This contradicts $dep(t_i, t_j, t_k)$. Additionally, the combinations of $dep(t_i, t_j, t_k)$ with $vital(t_i, t_k)$, $exc(t_k, t_j)$, and $exc(t_i, t_j)$ can also be expressed by only binary dependencies. The dependencies $exc(t_i, t_k)$, $vital\_dep(t_k, t_j)$ and $dep(t_k, t_j)$, on the other hand, makes the dependency $dep(t_i, t_j, t_k)$ superfluous. Here, only the commit of t_i and the abort of t_j, respectively, leads to an abort of t_k. The following binary dependencies can be combined with $dep(t_i, t_j, t_k)$:

$$X(t_i, t_k) : indep(t_i, t_k) \tag{8}$$

$$Y(t_k, t_j) : indep(t_k, t_j),\ vital(t_k, t_j) \tag{9}$$

$$Z(t_i, t_j) : indep(t_i, t_j),\ vital(t_i, t_j) \tag{10}$$

Due to dependency $exc(t_i, t_j, t_k)$, transaction t_k aborts in case both transactions t_i and t_j commit. Since a binary exclusive dependency between the related transactions already disallows that all transactions t_i, t_j, and t_k commit, the ternary dependency $exc(t_i, t_j, t_k)$ is superfluous. The following dependencies can be combined with $exc(t_i, t_j, t_k)$ without relaxing the latter dependency:

$$X(t_i, t_k) : indep(t_i, t_k),\ vital(t_i, t_k) \tag{11}$$

$$Y(t_k, t_j) : indep(t_k, t_j) \tag{12}$$

$$Z(t_i, t_j) : indep(t_i, t_j) \tag{13}$$

The combination of $exc(t_i, t_j, t_k)$ with the remaining binary dependencies can be expressed by only binary dependencies.

6 Example Transaction Closure

The following example is intended to clarify the application of transaction closures containing ternary termination dependencies. Especially, we show the combination of the ternary and binary termination dependencies in such a transaction closure. The transaction closure in our example can be considered as a workflow with special dependencies among the related transactions.

Example 2. A commonly used example is a travel planning activity. In our example this activity consists of reserving a flight, booking a room in the Hilton hotel including a diving course or booking a room in the Maritim hotel including a car rental. We model this activity as a transaction closure with the transactions t_2, t_3, t_4, t_5, t_6, and the coordinating root transaction t_1. Transaction t_2 represents the flight reservation which is essential for the trip. Moreover, a room in the Hilton hotel (t_3) or in the Maritim hotel (t_4) may be booked. The diving course (t_5) is directly connected with a room in the Hilton hotel. Additionally, we try to rent a car (t_6) which is a condition for staying at the Maritim hotel.

One possibility to model this scenario is as follows. The transactions t_2, t_3, and t_4 are child transactions of the root transaction t_1 and are connected to transaction t_1 by the following binary termination dependencies:

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

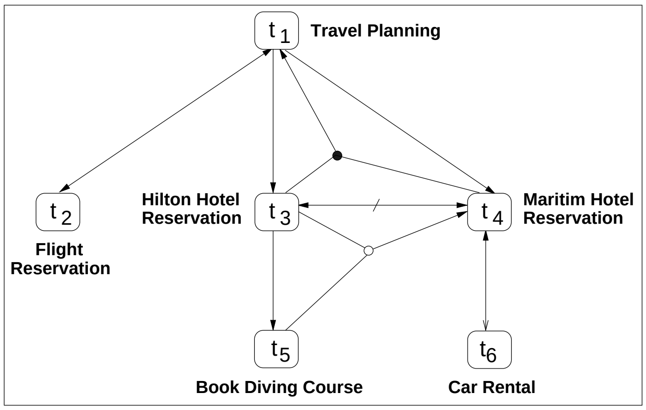
Transaction t_3 is vital for t_5 whereas transaction t_6 is vital-dependent on t_4:

$$vital(t_3, t_5) \ \wedge\ vital\_dep(t_4, t_6)$$

The travel planning activity has to be canceled if there is no room available in the hotels. Furthermore, there is the restriction that we only need a room in one of the hotels. These facts are expressed by the following dependencies:

$$vital(t_3, t_4, t_1) \ \wedge\ exc(t_3, t_5, t_4) \ \wedge\ exc(t_3, t_4)$$

Our example transaction closure is graphically illustrated as follows:



From our dependency definitions and the rules (5)–(13) we can now derive whether the specified ternary dependencies can be simplified to binary dependencies or not. We start with the ternary dependency $vital(t_3, t_4, t_1)$ and the corresponding transactions t_3, t_4, and t_1. The following dependencies are specified for $X(t_3, t_1)$, $Y(t_1, t_4)$, and $Z(t_3, t_4)$ (Here,we used the rule: $vital(t_i, t_j) \equiv dep(t_j, t_i)$):

$$dep(t_3, t_1) \wedge vital(t_1, t_4) \wedge exc(t_3, t_4)$$

Considering the rules (5)–(7) these dependencies are reasonable for $X(t_3, t_1)$, $Y(t_1, t_4)$, and $Z(t_3, t_4)$ in combination with $vital(t_3, t_4, t_1)$.

The ternary dependency $exc(t_3, t_5, t_4)$ has to be considered in combination with the dependencies $X(t_3, t_4)$, $Y(t_4, t_5)$, and $Z(t_3, t_5)$. If there is no dependency explicitly defined on these transaction pairs, we assume that the corresponding transactions are independent:

$$exc(t_3, t_4) \wedge indep(t_4, t_5) \wedge vital(t_3, t_5)$$

According to rule (11) the ternary dependency $exc(t_3, t_5, t_4)$ can be simplified when it is considered together with the dependency $exc(t_3, t_4)$. In this case, transaction t_4 aborts if transaction t_3 commits. However, the ternary exclusive dependency specifies that both transactions t_3 and t_5 have to commit to force the abort of t_4. Here, we have to decide whether $exc(t_3, t_4)$ or $exc(t_3, t_5, t_4)$ better fits the application semantics. In this example, the binary dependency is essential because only one of the room booking transactions should finish successfully. The diving course is only booked in addition to the room in the Hilton hotel. In consequence, the ternary exclusive dependency can be removed. □

Different ternary and binary termination dependencies allows the transaction designer to specify complex applications. Example 2 showed that we are able to reason about the dependencies of a transaction closure definition.

7 Conclusions and Outlook

Transaction closures are collections of transactions where the connection between the transactions is specified by special dependencies. In this paper, we extended the framework by ternary termination dependencies. The fact that n-ary termination dependencies can be expressed by combinations of binary and ternary dependencies make the problem of reasoning about the correctness of dependency combinations much more simple. In this context, we analyzed the relationship between ternary and binary dependencies. As a result, we stress out that some combinations are invalid or superfluous.

Currently we are investigating the transitivity of ternary termination dependencies. Here, we consider ternary as well as binary dependencies. For example, three transactions t_i, t_j and t_k are connected over a ternary dependency and one of them, e.g. t_i, is binary dependent on a forth transaction t_l. We are interested in the transitive binary dependencies between t_l and the transactions t_j and t_k. Our goal is to derive rules which can be used to automatically compute the transitive dependencies among arbitrary transactions of a transaction closure.

References

1. P. C. Attie, M. P. Singh, E. A. Emerson, A. Sheth, M. Rusinkiewicz. Scheduling Workflows by Enforcing Intertask Dependencies. *Distributed Systems Engineering*, 3(4):222–238, 1996.
2. A. Buchmann, M. T. Özsu, M. Hornick, D. Georgakopoulos, F. Manola. A Transaction Model for Active Distributed Object Systems. In [6], pp. 123–151.
3. P. K. Chrysanthis, K. Ramamritham. Synthesis of Extended Transaction Models Using ACTA. *ACM Transaction on Database Systems*, 19(3):450–491, 1994.
4. U. Dayal, M. Hsu, R. Ladin. A Transaction Model for Long-Running Activities. In G. M. Lohmann, A. Sernadas, R. Camps (eds.), *Proc. VLDB'91*, pp. 113–122. Morgan Kaufmann, 1991.
5. A. K. Elmagarmid (ed.). *Database Transaction Models For Advanced Applications.* Morgan Kaufmann, 1992.
6. D. Georgakopoulos, M. Hornick, A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.
7. M. Hsu, R. Ladin, D. R. McCarthy. An Execution Model For Active Data Base Management Systems. In C. Beeri, J. W. Schmidt, U. Dayal (eds.), *Proc. 3rd Int. Conf. on Data and Knowledge Bases: Improving Usability and Responsiveness, 1988*, pp. 171–179, Morgan Kaufmann, 1988.
8. J. E. B. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing.* MIT Press, Cambridge, MA, 1985.
9. M. Rusinkiewicz, W. Klas, T. Tesch, J. Wäsch, P. Muth. Towards a Cooperative Transaction Model — The Cooperative Activity Model. In U. Dayal, P. M. D. Gray, S. Nishio (eds.), *Proc. VLDB'95*, pp. 194–205, Morgan Kaufmann, 1995.
10. K. Schwarz, C. Türker, G. Saake. Analyzing and Formalizing Dependencies in Generalized Transaction Structures. In *Proc. Int. Workshop on Issues and Applications of Database Technology, IADT'98*, 1998.
11. K. Schwarz, C. Türker, G. Saake. Transitive Dependencies in Transaction Closures. In *Proc. Int. Database Engineering and Applications Symposium, IDEAS'98*, 1998.
12. J. Tang, J. Veijalainen. Enforcing Inter-task Dependencies in Transactional Workflows. In S. Laufmann, S. Spaccapietra, T. Yokoi (eds.), *Proc. 3rd Int. Conf. on Cooperative Information Systems, CoopIS'95*, pp. 72–86, 1995.