Global Transaction Termination Rules in Composite Database Systems

Can Türker[1], Kerstin Schwarz[2], and Gunter Saake[3]

[1] Swiss Federal Institute of Technology (ETH) Zürich
Institute of Information Systems, ETH Zentrum
CH-8092 Zürich, Switzerland
`tuerker@inf.ethz.ch`
[2] UBS AG
Flurstrasse 62
CH-8098 Zürich, Switzerland
`kerstin.schwarz@ubs.com`
[3] Otto-von-Guericke-Universität Magdeburg
Institut für Technische und Betriebliche Informationssysteme
Postfach 4120, D–39016 Magdeburg, Germany
`saake@iti.cs.uni-magdeburg.de`

Abstract. In composite database systems, global transactions are decomposed by the global transaction manager into several global subtransactions that are executed at the corresponding component database systems. This paper shows that the execution and termination of a global transaction depend on the specified extensional assertions on the local classes as well as on the given kinds of global requests. In some cases, a global transaction is supposed to be successful if exactly one global subtransaction is successfully executed. In other cases, all global subtransactions have to be successful in order to commit the global transaction. We discuss various termination rules for global transactions and thus provide new insights into a complex problem.

Keywords: global transactions, composite database systems, extensional assertions, termination dependencies.

1 Introduction

A *composite*[1] *database system* provides a homogeneous interface to possibly heterogeneous component database systems. This homogeneous interface consists of a global schema which is the result of a logical integration of the schemata of the corresponding component database systems [1, 15]. The integration is based on the *extensional assertions* [17, 19] which are specified by the database integrator.

[1] Here, we explicitly do not use the notion of *federated database systems* because we are interested in integration scenarios in which the component database systems are willing/cooperating to achieve global consistency. In contrast, as argued by many authors, the component database systems of a federated database system are assumed to be autonomous and thus not primarily interested in supporting global consistency.

These extensional assertions determine the relationships among the extensions of related local classes, e.g. extensional containment or equivalence. Extensional assertions as well as the global schema itself are stored in the global (meta) database managed by the composite database system.

Since these extensional assertions are integration constraints, they always have to be valid, i.e., the composite database system has to maintain these constraints; otherwise the global schema is based on false assumptions and thus it is incorrect. Using the global schema, *global transactions* are able to *transparently* access and manipulate data located in different component database systems [2]. Depending on the given global schema and particularly on the specified extensional assertions, a global transaction is decomposed into a set of *global subtransactions* that operate on the component database systems.

Figure 1 depicts a scenario in which two local classes Employee are integrated into one global class Employee. For simplicity, we assume that both classes have the same attributes. During schema integration the designer of the global schema has specified that the local classes are extensionally equivalent, i.e., these classes have to contain the same (semantically equivalent) objects in each database state. In consequence, a global insertion transaction t_g must be transformed into two global subtransactions t_1 and t_2 which try to perform the insertions in the corresponding component database systems. Due to the extensional assertion, both global subtransactions have to be successful in order to commit the global transaction. In case one global subtransaction commits while the other one aborts, the requirement that both classes are extensionally equivalent is violated. In order to correctly terminate a global transaction, a global transaction termination protocol is needed.

Besides extensional equivalence, there are further extensional assertions. Depending on the extensional assertions, different *global transaction termination rules* are needed to correctly terminate a global transaction. For instance, in some cases a commit of only one global subtransaction is sufficient to commit the global transaction, i.e., a global transaction can commit although some of the global subtransactions are aborted. In order to get a grasp of the meaningful global transaction termination rules, we analyze the underlying transaction termination dependencies. For that, we use the transaction closure framework [12, 13] which provides a complete set of the possible termination dependencies.

In particular, we show that a global transaction cannot always be executed by exactly one global subtransaction per corresponding component database system — which however is assumed by most approaches to transaction management in multidatabase systems. Based on our investigation of the effects of global insert, update, and delete transactions, we will show that in some cases several global subtransactions operating on the *same* component database system are required to correctly perform a global transaction.

The paper is organized as follows: Section 2 introduces transaction termination dependencies. In Section 3 we analyze termination dependencies of global transactions and derive meaningful global transaction termination rules. Section 4 discusses strategies for executing global transactions. Depending on the

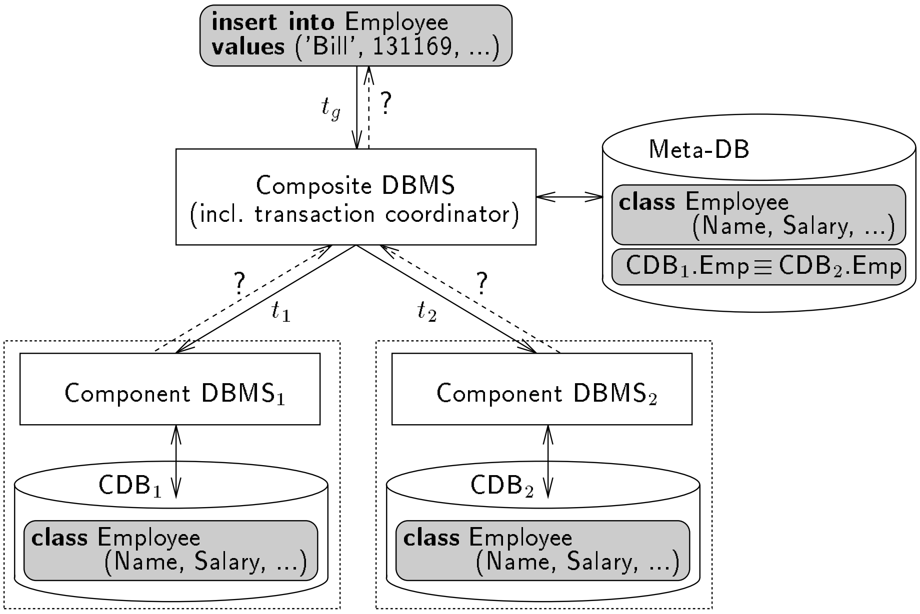

Fig. 1. Global Insertion into Extensionally Equivalent Classes

extensional assertions on the local classes, we investigate the underlying dependencies among the related transactions. In Section 5 we sketch a general protocol that captures the different global transaction termination rules. Section 6 concludes the paper.

2 Transaction Termination Dependencies

As formal framework for specifying and reasoning about dependencies between transactions we use the concept of a transaction closure which provides a complete set of termination dependencies [11, 13]. *Termination dependencies* are used to specify constraints on the occurrence of the significant termination events commit (c) and abort (a) of related transactions. For instance, if the global transaction aborts, then its global subtransactions have to abort, too.

We have identified a small set of binary and ternary termination dependencies being sufficient to specify any n-ary termination dependency among several transactions [12]. The binary dependencies are summarized in Table 2. For a dependency, a valid termination event combination of the related transactions is denoted by '✓' whereas the invalid ones are represented by '—'.

The termination dependency between the transactions t_i and t_j is **vital**(t_i, t_j) if the abort of t_i leads to the abort of t_j ("t_i is vital for t_j"). If this dependency holds in both directions, i.e., if the abort of t_i leads to the abort of t_j and vice versa, we say t_i and t_j are vital-dependent on each other, denoted **vital_dep**(t_i, t_j). The termination dependency **exc**(t_i, t_j) disallows that

	a_{t_i} a_{t_j}	a_{t_i} c_{t_j}	c_{t_i} a_{t_j}	c_{t_i} c_{t_j}
vital(t_i, t_j)	✓	—	✓	✓
vital_dep(t_i, t_j)	✓	—	—	✓
exc(t_i, t_j)	✓	✓	✓	—

Fig. 2. Binary Termination Dependencies

both transactions t_i and t_j commit together. In the default case, when no dependency is specified between two "independent" transactions, clearly all termination event combinations are valid. The formal specification of the termination dependencies is as follows ($\Rightarrow$ denotes implication):

$$\begin{aligned} \textbf{vital}(t_i, t_j) &:\Leftrightarrow (a_{t_i} \Rightarrow a_{t_j}) \\ \textbf{vital\_dep}(t_i, t_j) &:\Leftrightarrow \textbf{vital}(t_i, t_j) \wedge \textbf{vital}(t_j, t_i) \\ \textbf{exc}(t_i, t_j) &:\Leftrightarrow (c_{t_i} \Rightarrow a_{t_j}) \vee (c_{t_j} \Rightarrow a_{t_i}) \end{aligned}$$

There are termination dependencies among more than two transactions that cannot be expressed by conjunctions of binary dependencies alone. We identified three fundamental ternary dependencies that, in combination with the binary dependencies **vital** and **exc**, enable us to specify every possible n-ary termination dependency using conjunctive formulae only [12]. The ternary dependency **vital**(t_i, t_j, t_k) states that transaction t_k has to be aborted if both transactions t_i and t_j abort. The dependency **dep**(t_i, t_j, t_k) expresses that a commit of transaction t_i and an abort of transaction t_j leads to the abort of transaction t_k. In case of **exc**(t_i, t_j, t_k), the transaction t_k has to abort, if both transactions t_i and t_j commit:

$$\begin{aligned} \textbf{vital}(t_i, t_j, t_k) &:\Leftrightarrow (a_{t_i} \wedge a_{t_j}) \Rightarrow a_{t_k} \\ \textbf{dep}(t_i, t_j, t_k) &:\Leftrightarrow (c_{t_i} \wedge a_{t_j}) \Rightarrow a_{t_k} \\ \textbf{exc}(t_i, t_j, t_k) &:\Leftrightarrow (c_{t_i} \wedge c_{t_j}) \Rightarrow a_{t_k} \end{aligned}$$

As depicted in Table 3, ternary termination dependencies disallow exactly one termination event combination of the three related transactions.

	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)	✓	—	✓	✓	✓	✓	✓	✓
dep(t_i, t_j, t_k)	✓	✓	✓	✓	✓	—	✓	✓
exc(t_i, t_j, t_k)	✓	✓	✓	✓	✓	✓	✓	—

Fig. 3. Ternary Termination Dependencies

3 Global Transaction Termination Rules

Depending on the nature of a global transaction, different termination dependencies among the global transaction t_g and its global subtransactions t_1, ..., t_n are possible and meaningful. Nevertheless, the following general rules always hold:

Rule 1: Since a transaction cannot be forced to commit, the common abortion of all transactions should be always possible.

Rule 2: The abortion of the global transaction implies the abortion of all global subtransactions:

$$a_{t_g} \Rightarrow (a_{t_1} \wedge a_{t_2} \wedge \cdots \wedge a_{t_n})$$

This fact can be expressed by the following conjunction of binary dependencies:

$$\textsf{vital}(t_g, t_1) \wedge \textsf{vital}(t_g, t_2) \wedge \cdots \wedge \textsf{vital}(t_g, t_n)$$

Rule 3: The global transaction can only commit if at least one of the global subtransactions commits. In other words, the global transaction cannot commit if all global subtransactions abort:

$$c_{t_g} \Rightarrow (c_{t_1} \vee c_{t_2} \vee \cdots \vee c_{t_n})$$
$$\equiv (a_{t_1} \wedge a_{t_2} \wedge \cdots \wedge a_{t_n}) \Rightarrow a_{t_g}$$

This fact can be expressed by the following n-ary dependency:

$$\textsf{vital}(t_1, t_2, \ldots, t_n, t_g)$$

For brevity, but without loss of generality, our following considerations are restricted to a global transaction t_g with two global subtransactions t_1 and t_2. However, the results can easily be extended to the n-ary case.

Figure 4 represents the effects of the Rules 1–3 on the set of possible termination event combinations:

- The first rule states that each valid dependency must allow that all transactions may abort together. Thus, the first termination event combination must be always valid.
- The second rule forbids the termination event combinations where the global transaction t_g aborts and at least one of the global subtransactions t_1 or t_2 commits. Therefore, the second, third, and fourth termination event combinations are invalid.
- The third rule forbids the termination event combinations where the global transaction t_g commits and none of the global subtransactions t_1 or t_2 commit. Therefore, the fifth termination event combination is disallowed.

t_g	t_1	t_2	1	2	3	4	5	6	7	8	
a_{t_g}	a_{t_1}	a_{t_2}	✓	✓	✓	✓	✓	✓	✓	✓	Rule 1
a_{t_g}	a_{t_1}	c_{t_2}	—	—	—	—	—	—	—	—	Rule 2
a_{t_g}	c_{t_1}	a_{t_2}	—	—	—	—	—	—	—	—	Rule 2
a_{t_g}	c_{t_1}	c_{t_2}	—	—	—	—	—	—	—	—	Rule 2
c_{t_g}	a_{t_1}	a_{t_2}	—	—	—	—	—	—	—	—	Rule 3
c_{t_g}	a_{t_1}	c_{t_2}	✓	—	✓	✓	—	—	✓	—	
c_{t_g}	c_{t_1}	a_{t_2}	✓	✓	—	✓	—	✓	—	—	
c_{t_g}	c_{t_1}	c_{t_2}	✓	✓	✓	—	✓	—	—	—	

Fig. 4. Termination Event Combinations

Finally, three termination event combinations are free in that sense that they are not fixed by these rules. In consequence, there are eight possible dependency combinations which have to be considered.

Let us now discuss these eight cases which represent different *global transaction termination rules*:

1. **at least one**: This first case allows to commit the global transaction t_g if at least one of the global subtransactions t_1 or t_2 commits. That is, this global transaction termination rule does not require further restrictions in addition to that defined by the Rules 1-3:

$$\textbf{vital}(t_1, t_2, t_g) \wedge \textbf{vital}(t_g, t_1) \wedge \textbf{vital}(t_g, t_2)$$

2. **at least** t_1: Here, the global transaction t_g may only commit if the global subtransaction t_1 commits. This restriction is expressed by strengthening the dependency **vital**(t_g, t_1) to **vital_dep**(t_g, t_1). Altogether, the following dependencies are given:

$$\textbf{vital}(t_1, t_2, t_g) \wedge \textbf{vital\_dep}(t_g, t_1) \wedge \textbf{vital}(t_g, t_2)$$

3. **at least** t_2: In contrast to the previous case, here the global transaction t_g may only commit if the global subtransaction t_2 commits. This restriction is expressed by strengthening the dependency **vital**(t_g, t_2) to **vital_dep**(t_g, t_2):

$$\textbf{vital}(t_1, t_2, t_g) \wedge \textbf{vital}(t_g, t_1) \wedge \textbf{vital\_dep}(t_g, t_2)$$

4. **exactly one**: In this case, the global transaction t_g may only commit if exactly one of the global subtransactions t_1 or t_2 commits. Thus, it is not allowed that both global subtransactions commit together. This condition is expressed by an additional dependency **exc**(t_1, t_2):

$$\textbf{vital}(t_1, t_2, t_g) \wedge \textbf{vital}(t_g, t_1) \wedge \textbf{vital}(t_g, t_2) \wedge \textbf{exc}(t_1, t_2)$$

5. **all**: This termination rule is well-known from distributed database systems. The global transaction t_g may only commit if all global subtransactions

commit. In other words, if one transaction aborts, then the other transactions have to abort, too. This fact is described by restricting the dependencies **vital**(t_g, t_1) and **vital**(t_g, t_2) to the dependencies **vital_dep**(t_g, t_1) and **vital_dep**(t_g, t_2), respectively:

$$\textbf{vital}(t_1, t_2, t_g) \wedge \textbf{vital\_dep}(t_g, t_1) \wedge \textbf{vital\_dep}(t_g, t_2)$$

Note that the ternary dependency is implied by the binary dependencies.

6. **only** t_1: The global subtransaction t_2 does not need to be started because it always has to abort. Thus, the consideration can be reduced to the global transaction t_g and the global subtransaction t_1 that are connected by the dependency **vital_dep**(t_g, t_1).
7. **only** t_2: The global subtransaction t_1 does not need to be started because it always has to abort. Therefore, the consideration can be reduced to the global transaction t_g and the global subtransaction t_2 that are connected by the dependency **vital_dep**(t_g, t_2).
8. **None**: This case is not meaningful since all transactions always have to abort.

Note that the **at least one** rule corresponds to the "weakest" dependency combination. The other dependency combinations restrict this dependency combination either by strengthening some of the dependencies or by adding further dependencies. Also, note that the **only** rules corresponds to the **all** rule on a reduced set of related transactions. Here, the involved transactions either commit or abort together.

4 Relationships between Extensional Assertions and Global Transactions Termination Rules

In this section, we consider global insert, update, and delete transactions. Again, we consider a scenario in which the composite database system relies on two component database systems. A global transaction t_g is executed on a global class C_g that is composed by the union of the local classes C_1 and C_2. This global transaction is decomposed into two global subtransactions t_1 and t_2. The global subtransaction t_1 works on the local class C_1 whereas the global subtransaction t_2 operates on the local class C_2 (of another database system). The termination dependencies between these transactions depend on the extensional assertions on the local classes as well as on the global update strategy.

4.1 Extensional Assertions on Classes

The *extension* of a class is the set of instances of this class in a given database state. $Ext^S_{C_1}$ denotes the extension of the class C_1 in a database state S. Let C_1 and C_2 be two semantically related classes. Then, the database integrator may define one of the following binary *extensional assertions* [17, 20] on the classes C_1 and C_2:

$$\begin{aligned}
\textit{Disjointness}:&\quad C_1 \varnothing C_2 \;:\Leftrightarrow\; \forall S\colon Ext^S_{C_1} \cap Ext^S_{C_2} = \emptyset\\
\textit{Equivalence}:&\quad C_1 \equiv C_2 \;:\Leftrightarrow\; \forall S\colon Ext^S_{C_1} = Ext^S_{C_2}\\
\textit{Containment}:&\quad C_1 \supset C_2 \;:\Leftrightarrow\; \forall S\colon Ext^S_{C_1} \supseteq Ext^S_{C_2}\\
\textit{Overlap}:&\quad C_1 \Cap C_2 \;:\Leftrightarrow\; \textit{true}
\end{aligned}$$

Two classes are extensionally disjoint if there are no common objects in the extension of these classes in any database state. Extensionally equivalent classes have the same extensions in each database state. One class extensionally contains another class if the extension of the first class is a superset of the extension of the other class in any database state. Finally, in case of overlap there are no restrictions on the relationship between the extensions of the two related classes.

4.2 Global Insert Transactions

From the global point of view, an object can be seen as (globally) inserted if at least one of the global subtransactions is successful, i.e., the object is inserted into at least one local class. In this case, the object is inserted into the composite database. However, this insertion must satisfy the extensional assertions specified on the related local classes. Since in general it is not known in advance in which local class(es) the globally created object may be inserted, for each corresponding local class an insertion operation has to be started.

Insert into $C_g = C_1 \cup C_2$ with $C_1 \varnothing C_2$

In case of extensionally disjoint classes exactly one of the global subtransactions is allowed to commit. Thus, in this scenario the **exactly one** rule has to be applied for the transactions t_g, t_1, and t_2.

However, if the global class contains a *discriminant attribute* [3] whose value corresponds to an individual local class, then only one global subtransaction t_i is started to perform the insertion. In this case, the **only** t_i rule is used.

Insert into $C_g = C_1 \cup C_2$ with $C_1 \equiv C_2$

Extensionally equivalent classes require that a globally created object is inserted either into all related local classes or into none of them. From this follows that the abort of one of the transactions t_g, t_1, and t_2 leads to the abort of the other transactions. The **all** rule corresponds to this scenario.

Insert into $C_g = C_1 \cup C_2$ with $C_1 \supset C_2$

In this case, an insertion into class C_2 requires that the same object is member of the class C_1. On the other hand, if an object is inserted into class C_1, the object must not necessarily be inserted into class C_2. Thus, different insertion strategies are possible:

- The strongest requirement on the global level is called the *maximum strategy*. This strategy requires that a globally created object has to be inserted into all related local classes. We can guarantee this requirement only by using the **all** rule.
- Following the *flexible strategy*, the globally created object has to be inserted into as many local classes as possible. However, the containment assertion requires that the object has to be inserted at least into class C_1. Inserting the object into class C_2 is optional. This refers to the **at least** t_1 rule.
- The *minimum strategy* inserts the globally created object using a minimal number of global subtransactions such that the extensional relationships between the local classes are maintained. Since the class C_1 extensionally contains the class C_2, the object has to be inserted only into class C_1. The subtransaction t_2 does not need to be executed. This corresponds to the **only** t_1 rule.

Insert into $C_g = C_1 \cup C_2$ with $C_1 \Cap C_2$

In case of extensionally overlapping classes the globally created object may be inserted into only one or both related local classes. As above, different insertion strategies are possible:

- The *maximum strategy* requires that the globally created object is inserted into all related local classes. Thus, we have to use the **all** rule.
- The *flexible strategy* tries to insert the globally created object into as many local classes as possible. Thus, all global subtransactions are submitted to the local systems. In order to commit the global transaction, at least one of them has to finish successfully. In case all global subtransactions abort, the object is not inserted. This strategy corresponds to the **at least one** rule.
- The *minimum strategy* requires that the globally created object is inserted by a minimal number of global subtransactions. For instance, in case of extensionally including classes the object is only inserted into the class which extensionally contains the other class. For extensionally overlapping classes the insertion can be done in both classes. Therefore, one of the candidate classes has to be chosen:
  - *Statically*: The administrator of the composite database system identifies the class during schema integration.
  - *Dynamically*: Depending on different values, such as the load of the system or the amount of local objects, the global transaction coordinator dynamically chooses a local class.
  - *Randomly*: The global transaction coordinator randomly chooses one of the local classes.

In all cases, only one of the global subtransactions is initiated to execute the global transaction. The other global subtransaction is not started. This refers to the **only** t_i rule.

– Finally, we can use a mixture of the strategies above which is called *flexible strategy with priorities*. In this case, we follow the flexible strategy, and thus insert the globally created object into as many classes as possible. Furthermore, one of the classes obtains priority. This means that the object has to be inserted into this class. As described above, the priority can be given by a static, dynamic, or random method. For example, if the priority is given to class C_1, then the related transaction t_1 has to be executed successfully. This refers to the **at least** t_1 rule. If class C_2 has priority, the system must follow the **at least** t_2 rule.

4.3 Global Update and Delete Transactions

Objects may also be globally updated or deleted. An object is considered as globally updated or deleted, respectively, if all corresponding objects in the local classes are updated or deleted. However, we cannot always deduce from the extensional assertions in which classes the object to be updated or deleted participates. For example, in extensionally overlapping classes the object may be in one of the classes or in both of them. Due to possible local insert operations submitted by local transactions, we even cannot conclude from the global insert strategy in which local classes an object appears. Thus, we always have to submit both global subtransactions t_1 and t_2 to the local systems. A global subtransaction aborts only if the object to be updated or deleted is member of the related classes but cannot be updated or deleted successfully. We follow the **all** rule to successfully update and delete global objects. A transaction that does not find an object to update or delete is considered as executed successfully. Otherwise, we could not deduce the correct reason for the abortion of a global subtransaction.

Figure 5 summarizes the rules for global inserts, updates, and deletes.

	at least one	**at least** t_1	**at least** t_2	**exactly one**	**all**	**only** t_1	**only** t_2
insert ($C_1 \emptyset C_2$)	—	—	—	✓	—	—	—
insert ($C_1 \equiv C_2$)	—	—	—	—	✓	—	—
insert ($C_1 \supset C_2$)	—	✓	—	—	✓	✓	—
insert ($C_1 \Cap C_2$)	✓	✓	✓	✓	✓	✓	✓
update / delete	—	—	—	—	✓	—	—

Fig. 5. Termination Rules for Global Operations

4.4 Combining Transaction Termination Rules

In the previous subsections we have considered global transactions with a single operation. However, transactions usually consist of several operations. These operations may also concern different global classes.

Traditional approaches [2, 7, 5] supporting global transactions in federated or multidatabase systems assume that for each global transaction there is *at most one* global subtransaction per component database system. In the following, we show that under this assumption certain kinds of global transactions cannot be executed (terminated) correctly.

As a counterexample we consider the following scenario. The global transaction t_g consists of the following insert and delete operations (where the classes C_1 and C_3 are managed by one component database system while the classes C_2 and C_4 are managed by another component database system):

Insert into $C_{g_1} = C_1 \cup C_2$ with $C_1 \varnothing C_2$;
Delete from $C_{g_2} = C_3 \cup C_4$ with $C_3 \equiv C_4$;

The global transaction is decomposed into two global subtransactions t_1 and t_2. The transaction t_1 performs the insertion into the local class C_1 and deletion from the local class C_3 whereas the transaction t_2 executes the insertion into the local class C_2 and the deletion from the local class C_4. The question is now:

In which cases can the global transaction be said to have executed successfully? If both global subtransactions commit or if exactly one global subtransaction commits?

A commit of both global subtransactions means that the insertion is performed on the local classes C_1 and C_2 and the deletion is executed on the local classes C_3 and C_4. However, in this case the extensional relationship between the classes C_1 and C_2 is not disjoint anymore, i.e., the extensional assertion cannot be guaranteed. Therefore, both global subtransactions cannot commit together. On the other hand, if both global subtransactions cannot commit together, the deletion cannot be executed successfully.

Thus, under the usual assumption that global subtransactions do not communicate with each other, this global transaction cannot be performed by two global subtransactions only.

A possible solution to the example discussed above is to initiate two global subtransactions for each global operation. The global subtransactions t_1 and t_2 execute the local insert operations whereas the global subtransactions t_3 and t_4 perform the local delete operations. According to the rules presented in Figure 5, the insertion transactions are terminated using the **exactly one** rule whereas the deletion transactions are terminated using the **all** rule.

The ternary dependency between the transactions t_g, t_3, and t_4 is eliminated because it is already expressed by the binary dependencies **vital_dep**(t_g, t_3) and

$\textbf{vital\_dep}(t_g, t_4)$. In summary, we have the following dependencies:

$$\begin{aligned} & c_{t_g} \Rightarrow ((c_{t_1} \wedge a_{t_2}) \vee (a_{t_1} \wedge c_{t_2})) \wedge c_{t_3} \wedge c_{t_4} \\ \equiv\ & (a_{t_1} \wedge a_{t_2}) \vee (c_{t_1} \wedge c_{t_2}) \vee a_{t_3} \vee a_{t_4} \Rightarrow a_{t_g} \\ \equiv\ & \textbf{vital}(t_1, t_2, t_g) \wedge \textbf{vital}(t_g, t_1) \wedge \textbf{vital}(t_g, t_2) \wedge \\ & \textbf{exc}(t_1, t_2) \wedge \textbf{vital\_dep}(t_g, t_3) \wedge \textbf{vital\_dep}(t_g, t_4) \end{aligned}$$

Thus, here we have a mixture of the rules **exactly one** and **all**. In the sequel, the generalization of this rule will be denoted as the $\textbf{mixed}_i$ rule. The index i states that only one of the transactions $t_1, ..., t_i$ and all of the transactions $t_{i+1}, ..., t_n$ have to commit in order to commit the global transaction t_g. More formally:

$$\textbf{mixed}_i(t_1, ..., t_n, t_g) :\Leftrightarrow (c_{t_g} \Leftrightarrow \biguplus_{j=1}^{i} c_{t_j} \wedge \bigwedge_{k=i+1}^{n} c_{t_k}), \text{ where}$$

$$\biguplus_{j=1}^{i} c_{t_j} :\Leftrightarrow (c_{t_1} \wedge a_{t_2} \wedge ... \wedge a_{t_i}) \vee (a_{t_1} \wedge c_{t_2} \wedge ... \wedge a_{t_i}) \vee ... \vee (a_{t_1} \wedge a_{t_2} \wedge ... \wedge c_{t_i})$$

As depicted in Figure 5, each global operation corresponds to a global rule. Global operations can be combined into one global subtransaction if the corresponding rules are compatible. In Figure 6 the combinations of all rules are summarized. Due to space restrictions, we use abbreviations: **a** stands for **at least**, **e** for **exactly one**, and **o** for **only**.[2]

	a one	**a** t_1	**a** t_2	**e one**	**all**	**o** t_1	**o** t_2
a one	**a one**	**a** t_1	**a** t_2	**e one**	**all**	**o** t_1	**o** t_2
a t_1	**a** t_1	**a** t_1	**a** $t_1 \wedge t_2$	**o** t_1	**all**	**o** t_1	**a** $t_1 \wedge t_2$
a t_2	**a** t_2	**a** $t_1 \wedge t_2$	**a** t_2	**o** t_2	**all**	**a** $t_1 \wedge t_2$	**o** t_2
e one	**e one**	**o** t_1	**o** t_2	**e one**	—	**o** t_1	**o** t_2
all	**all**	**all**	**all**	—	**all**	**all**	**all**
o t_1	**o** t_1	**o** t_1	**a** $t_1 \wedge t_2$	**o** t_1	**all**	**o** t_1	**o** $t_1 \wedge t_2$
o t_2	**o** t_2	**a** $t_1 \wedge t_2$	**o** t_2	**o** t_2	**all**	**o** $t_1 \wedge t_2$	**o** t_2

Fig. 6. Combining Transaction Termination Rules

For brevity, we do not discuss all combinations. Rather we mention some general rules and interesting cases:

[2] **a** $t_1 \wedge t_2$ stands for **at least** t_1 **and** t_2; analogously **o** $t_1 \wedge t_2$ stands for **only** t_1 **and** t_2. In our case — we only consider two global subtransactions t_1 and t_2 — these rules correspond to the **all** rule. The distinction between these rules becomes essential when considering more than two global subtransactions.

- The **at least one** rule represents the weakest requirement, i.e., this rule is always strengthened by another one. For example, **at least one** and **exactly one** are combined to **exactly one**.

  Please note that this strengthening may lead to termination rules that are too strong in some cases — these rules are depicted by light gray colored boxes in Figure 6. Suppose, there is a global transaction with the following two operations:

  Insert into $C_{g_1} = C_1 \cup C_2$ with $C_1 \Cap C_2$ using the flexible strategy;
  Insert into $C_{g_2} = C_3 \cup C_4$ with $C_3 \Cap C_4$ using the flexible strategy;

  Both operations correspond to the **at least one** rule. However, the following situation may occur: In the global subtransaction t_1, the insertion into C_1 fails while the insertion into C_3 could be performed successfully if the former operation would have not been failed. In the global subtransaction t_2, the insertion into C_4 fails while the insertion into C_2 could be performed successfully if the former operation would have not been failed. In consequence, both global subtransactions, and thus the global transaction, fail.

  Consequently, a global transaction may fail although, from a global point of view, there is a way to successfully perform the operations of this transaction.

- Combining **at least** t_i with **exactly one** results in **only** t_i because in this case the global transaction can only commit if t_i commits. However, if t_i commits, then the other global subtransaction has to abort.

- The **all** rule depicts the strongest requirement, i.e., a combination with the **all** rule always leads to the **all** rule. One exception is the incompatible combination **all** and **exactly one** (which was already discussed by the example before). This incompatibility has a fundamental consequence:

  The corresponding global operations must be performed by different global subtransactions.

- The rules **all** and **only** t_1 can be combined if the global subtransaction t_1 contains both operations whereas the global subtransaction t_2 only contains the operation corresponding to the **all** rule. To illustrate this case, suppose that the global transaction consists of the following two operations:

  Insert into $C_{g_1} = C_1 \cup C_2$ with $C_1 \equiv C_2$;
  Insert into $C_{g_2} = C_3 \cup C_4$ with $C_3 \supset C_4$ using the minimal strategy;

  The first operation requires the **all** rule whereas the second one needs the **only** t_1 rule (under the assumption that the global subtransaction t_1 performs the insertion into the local class C_3). The global transaction can be performed by two global subtransactions t_1 and t_2 such that t_1 executes the insertions into the local classes C_1 and C_3 and t_2 performs the insertion into the local class C_2. The global transaction may only commit if both global subtransactions commit. Thus, the resulting rule is **all**.

The dark gray colored boxes in Figure 6 refer to such kinds of combinations where the global operation is not executed in both global subtransactions, i.e., using the minimal semantics.

In summary, we can state that the combinations in the "white" and dark gray colored boxes (except the ones with '—') are the "good" ones which always satisfy the given dependencies and perform all global operations in a correct way. In contrast, the combinations in the light gray colored boxes are the "potentially too strong" ones. These combinations might lead to executions that are rejected due to a violation of the given dependencies although there is an execution that would satisfy the given dependencies and perform all global operations in a correct way. Note that these combinations unfold a fundamental problem of executing global transactions under dependencies: To avoid unnecessary rejections of a global transaction, all possible executions have to be considered in the worst case.

5 Global Transaction Termination Protocol

Atomic commitment protocols such as the *two-phase commit (2PC)* protocol [6] and its variants *presumed abort/commit* [8] have been studied extensively in the context of distributed database systems. As pointed out in [9], the problem of the atomic commitment in federated database systems cannot be solved without locally participating in an atomic commitment protocol. There are several approaches to ensure global transaction atomicity, e.g. [10, 16, 7, 14, 21, 18, 4]. All these approaches focus on *atomic commitment* of global transactions. As we have shown in this paper, global atomic commitment alone is not sufficient to correctly deal with the termination of global transactions.

Following our previous discussion, a global transaction termination rule corresponds to a boolean formula. As we have seen before, atomic commitment is a special case. Therefore, a generalization of the two-phase-commit (2PC) protocol is needed to capture the various transaction termination rules. Figure 7 sketches such a global transaction termination protocol (that does not consider site failures). The circles denote the states and the dashed lines indicate messages between the *coordinator* (global transaction) and the *participants* (global subtransactions).

The traditional 2PC protocol has been modified as follows (the modified parts are denoted by the thick lines in Figure 7). In the **WAIT** state, originally the coordinator waits for the votes from *all* participants. In the generalized protocol, the coordinator evaluates the formula corresponding to the given termination rule immediately after having received a vote from a participant. As soon as the formula yields true, the coordinator decides for a "global commit". A global commit does not necessarily imply that all participants that have voted for commit will receive a global commit message. For instance, in case of the **exactly one** rule, only one participant may commit even if several participants may have voted for commit.

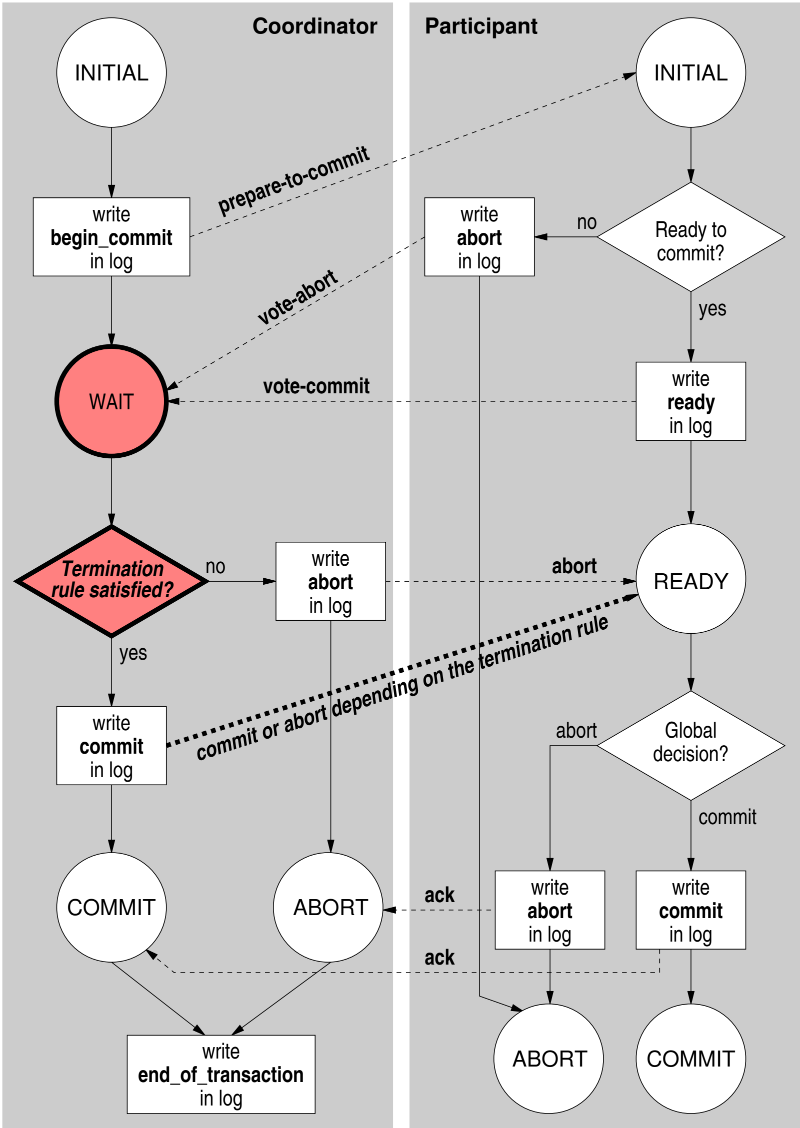

Fig. 7. The Generalized Transaction Termination Protocol

Note that the participant part of the 2PC protocol does not need to be modified. A participant that has voted for commit remains in the state **READY** until it receives the global decision message. In contrast to the traditional 2PC protocols, here a participant cannot infer from a commit message that all other participants have committed, too. Analogously, an abort message does not imply that no other participants has committed.

6 Conclusions

The contributions of this paper can be summarized as follows:

1. We have systematically investigated possible and meaningful global transaction termination rules and their correspondence to global operations. Especially, we pointed out the impact of extensional assertions on the correct termination strategy for global transactions. We have shown that atomic commitment is not necessary in all cases. In the **exactly one** case atomic commitment is even the wrong choice because only one global subtransaction is allowed to commit in order to commit the global transaction.
2. Moreover, we have shown that there are cases in which it is not possible to (correctly) perform a global transaction by executing at most one global subtransaction per component database system. This, however, is the general assumption on which most approaches to transaction management in composite (federated) database systems rely on. From our analysis we can conclude which operations can be combined within a global subtransaction.
3. Finally, we sketched a generalized protocol that supports the various global transaction termination rules. As an important issue, this protocol requires only changes of the 2PC coordinator protocol. Therefore, this generalized protocol can be used in a composite database system environment (without further restricting the local autonomy) if the component database systems support the standard 2PC participant protocol.

The knowledge about the correct decomposition and termination of global transactions is essential for maintaining globally consistent databases. This also includes that local transactions have to observed and necessary actions triggered when extensional assertions are violated. In this way, a local transaction becomes a global one.

Acknowledgments: Thanks to Ingo Schmitt for helpful comments on an earlier draft of this paper. We are also thankful to the anonymous referees for their valuable suggestions.

References

1. C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.
2. Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of Multidatabase Transaction Management. *The VLDB Journal*, 1(2):181–240, October 1992.
3. S. Conrad, I. Schmitt, and C. Türker. Considering Integrity Constraints During Federated Database Design. In S. M. Embury, N. J. Fiddian, A. W. Gray, and A. C. Jones, editors, *Advances in Databases, 16th British National Conf. on Databases, BNCOD 16, Cardiff, Wales, July 1998*, Lecture Notes in Computer Science, Vol. 1405, pages 119–133. Springer-Verlag, Berlin, 1998.

4. A. K. Elmagarmid, J. Jing, W. Kim, O. Bukhres, and A. Zhang. Global Committability in Multidatabase Systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(5):816–824, October 1996.
5. D. Georgakopoulos, M. Rusinkiewicz, and A. Sheth. Using Tickets to Enforce the Serializability of Multidatabase Transactions. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):166–180, February 1994.
6. J. Gray. Notes on Data Base Operating Systems. In R. Bayer, R. M. Graham, and G. Seegmüller, editors, *Operating Systems, An Advanced Course*, Lecture Notes in Computer Science, Vol. 60, pages 393–481. Springer-Verlag, Berlin, 1978.
7. S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth, and A. Silberschatz. Ensuring Transaction Atomicity In Multidatabase Systems. In *Proc. of the 11th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS'92, San Diego, CA*, pages 164–175. ACM Press, New York, 1992.
8. C. Mohan, B. Lindsay, and R. Obermarck. Transaction Management in the R* Distributed Database Management System. *ACM Transaction on Database Systems*, 11(4):378–396, December 1986.
9. J. G. Mullen, A. K. Elmagarmid, W. Kim, and J. Sharif-Askary. On the Impossibility of Atomic Commitment in Multidatabase Systems. In P. A. Ng, C. V. Ramamoorthy, L. C. Seifert, and R. T. Yeh, editors, *Proc. of the Second Int. Conf. on System Integration, ICSI'92*, pages 625–634. IEEE Computer Society Press, Los Alamitos, CA, 1992.
10. P. Muth and T. Rakow. Atomic Commitment for Integrated Database Systems. In N. Cercone and M. Tsuchiya, editors, *Proc. of the 7th IEEE Int. Conf. on Data Engineering, ICDE'91, April 8–12, 1991, Kobe, Japan*, pages 296–304. IEEE Computer Society Press, Los Alamitos, CA, 1991.
11. K. Schwarz. *The Concept of Transaction Closure for the Consistent Specification of Dependencies in Complex Applications*, Dissertationen zu Datenbanken und Informationssystemen, Vol. 65. infix-Verlag, Sankt Augustin, 1999. (In German).
12. K. Schwarz, C. Türker, and G. Saake. Extending Transaction Closures by N-ary Termination Dependencies. In W. Litwin, T. Morzy, and G. Vossen, editors, *Advances in Databases and Information Systems, Proc. Second East-European Symposium, ADBIS'98, Poznań, Poland, September 1998*, Lecture Notes in Computer Science, Vol. 1475, pages 131–142. Springer-Verlag, Berlin, 1998.
13. K. Schwarz, C. Türker, and G. Saake. Integrating Execution Dependencies into the Transaction Closure Framework. *International Journal of Cooperative Information Systems (IJCIS)*, 8(2–3):111–138, June/September 1999.
14. A. Sheth, M. Rusinkiewicz, and G. Karabatis. Using Polytransaction to Manage Interdependent Data. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 555–581, Morgan Kaufmann Publishers, San Mateo, CA, 1992.
15. A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.
16. N. Soparkar, H. Korth, and A. Silberschatz. Failure-Resilient Transaction Management in Multidatabases. *IEEE Computer*, 24(12):28–36, December 1991.
17. S. Spaccapietra, C. Parent, and Y. Dupont. Model Independent Assertions for Integration of Heterogeneous Schemas. *The VLDB Journal*, 1(1):81–126, July 1992.
18. A. Tal and R. Alonso. Integration of Commit Protocols in Heterogeneous Databases. *Distributed and Parallel Databases*, 2(2):209–234, 1994.

19. C. Türker. *Semantic Integrity Constraints in Federated Database Schemata*, Dissertationen zu Datenbanken und Informationssystemen, Vol. 63. infix-Verlag, Sankt Augustin, 1999.
20. C. Türker and G. Saake. Consistent Handling of Integrity Constraints and Extensional Assertions for Schema Integration. In J. Eder, I. Rozman, and T. Welzer, editors, *Advances in Databases and Information Systems, Proc. Third East-European Symposium, ADBIS'99, Maribor, Slovenia, September 1999*, Lecture Notes in Computer Science, Vol. 1691, pages 31–45. Springer-Verlag, Berlin, 1999.
21. J. Veijalainen and A. Wolski. Prepare and Commit Certification for Decentralized Transaction Management in Rigorous Heterogeneous Multidatabases. In F. Golshani, editor, *Proc. of the 8th IEEE Int. Conf. on Data Engineering, ICDE'92, Tempe, Arizona, USA, February 2–3, 1992*, pages 470–480. IEEE Computer Society Press, Los Alamitos, CA, 1992.