Transaction Dependencies in Generalized Transaction Structures (Abstract)

Kerstin Schwarz   Can Türker   Gunter Saake

Institut für Technische Informationssysteme
Otto-von-Guericke-Universität Magdeburg
Postfach 4120, D–39016 Magdeburg, Germany
E-mail: {schwarz|tuerker|saake}@iti.cs.uni-magdeburg.de
Phone: ++49/391/67-{18066|12994|18800}   Fax: ++49/391/67-12020

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

The concept of *transaction closure* as a generalization of the well-known concept of nested transactions [Mos85] together with a set of transaction dependencies provide a common framework for modeling advanced, complex transactions/activities. A transaction closure comprises of a set of transactions which are transitively initiated by the same (root) transaction. In contrast to nested transactions, a child transaction in a transaction closure may survive the termination of its parent transaction — a case which is needed for example in models for long-during activities [DHL91, RKT⁺95], workflows [GHS95, KR96], or in active databases [HLM88, BÖH⁺92].

Our goal is to find a (minimal) set of (orthogonal) fundamental transaction dependencies which are applicable according to real-world application semantics. For that, we investigate termination, execution order and object visibility dependencies. A termination dependency is a constraint on the possible combinations (and orders) of the termination events (commit/abort) of two related transactions. In comparison to nested transactions, in transaction closures the framework for termination dependencies has to be extended, e.g. a child transaction may leave the scope of its parent transaction. Whereas the termination condition for direct child transactions are explicitly specified, the conditions for transitively related transactions have to be computed based on the direct dependencies. We identify five reasonable termination dependencies (see Table 1). A ($\surd$) denotes the corresponding termination event combination is valid for the specified dependency. Otherwise, in the case of (–) the event combination is not allowed to occur.

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

Table 1: Termination Dependencies between two Transactions t_i and t_j

Additionally to the termination dependencies, there are execution order dependencies which specifies whether a transaction is executed in parallel or sequential to another transaction. In other words, there exist constraints on the termination event and the begin event of the related transactions. Object visibility constraints influences the view of transactions. On the one hand the state of objects accessed by the parent transaction are make visible to a child with its initiation. On the other hand, effects are made visible with the commit of a transaction to another transaction or to all other transactions. These dependencies may also influence the termination dependencies among transactions.

In particular, we consider the transitivity property for all kinds of transaction dependencies discussed above. For example, the abortion of a transaction may lead to the abortion of parts of the whole transaction closure. Figure 1 illustrates termination dependencies between pairs of transactions ($t_i \rightarrow t_j$ means that: the abortion of t_i leads to the abortion of t_j) and the derived transitive dependencies among the transactions (represented by dashed arrows). In the example represented in Figure 1, the abortion of transaction t_3 implicitly leads to the abortion of transaction t_1.

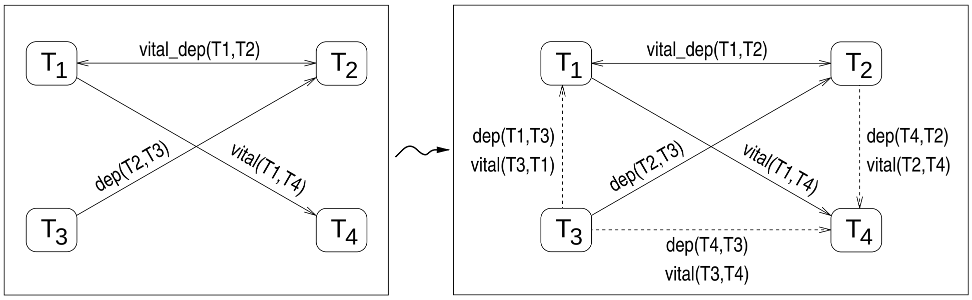

Figure 1: Example for Transitive Transaction Dependencies

Using the transitive relationships, we are able to derive the exact relationship between two arbitrary transactions. This issue is essential to detect hidden (not explicitly specified) relationships among transactions as well as failures and contradictions in the specification of a transaction closure during the design process. As a result, the concept of transaction closure and the basic dependencies provide the basis for a transaction design and analyzing tool. Such a tool can help to understand the entire semantics of a complex application and thus it may support the design of better and more efficient applications, particularly in multidatabase systems or in workflow systems. The concepts of a transaction closure and of transaction dependencies with their transitivity properties are formally defined by using the ACTA framework [CR90, CR94].

References

[BÖH+92] A. Buchmann, M. T. Özsu, M. Hornick, D. Georgakopoulos, and F. Manola. A Transaction Model for Active Distributed Object Systems. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 123–151, Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[CR90] P. K. Chrysanthis and K. Ramamritham. ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior. In H. Garcia-Molina and H. V. Jagadish, editors, *Proc. of the 1990 ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ*, ACM SIGMOD Record, Vol. 19, No. 2, pages 194–203, June 1990.

[CR94] P. K. Chrysanthis and K. Ramamritham. Synthesis of Extended Transaction Models Using ACTA. *ACM Transaction on Database Systems*, 19(3):450–491, September 1994.

[DHL91] U. Dayal, M. Hsu, and R. Ladin. A Transaction Model for Long-Running Activities. In G. M. Lohmann, A. Sernadas, and R. Camps, editors, *Proc. of the 17th Int. Conf. on Very Large Data Bases (VLDB'91), Barcelona, Spain*, pages 113–122. Morgan Kaufmann Publishers, San Mateo, CA, September 1991.

[GHS95] D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.

[HLM88] M. Hsu, R. Ladin, and D. R. McCarthy. An Execution Model For Active Data Base Management Systems. In C. Beeri, J. W. Schmidt, and U. Dayal, editors, *Proc. of the 3rd Int. Conf. on Data and Knowledge Bases: Improving Usability and Responsiveness, Jerusalem, Israel, June, 1988*, pages 171–179, Morgan Kaufmann Publishers, 1988.

[KR96] M. U. Kamath and K. Ramamritham. Correctness Issues in Workflow Management. *Distributed Systems Engineering*, 3(4), December 1996.

[Mos85] J. E. B. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing.* MIT Press, Cambridge, MA, 1985.

[RKT+95] M. Rusinkiewicz, W. Klas, T. Tesch, J. Wäsch, and P. Muth. Towards a Cooperative Transaction Model — The Cooperative Activity Model. In U. Dayal, P. M. D. Gray, and S. Nishio, editors, *Proc. of the 21st Int. Conf. on Very Large Data Bases (VLDB'95), Zürich, Switzerland*, pages 194–205, Morgan Kaufmann Publishers, September 1995.