VODAK Open Nested Transactions – Visualizing Database Internals

Peter Muth and Thomas C. Rakow

GMD – Integrated Publication and Information Systems Institute (IPSI)
D-W6100 Darmstadt, Germany
{muth,rakow}@darmstadt.gmd.de

1. Description

VODAK is a prototype of an object-oriented, distributed database system developed during the past five years at the Integrated Publication and Information Systems Institute (IPSI). The aim of demonstrating VODAK Open Nested Transactions is to provide insights into internals of database systems that are usually hidden from application programmers and users. By utilizing semantics of methods, VODAK Open Nested Transactions increase the degree of parallelism between concurrent transactions compared to conventional transaction management schemes. Demonstrating the difference in parallelism provides users with a "feeling" for internal database mechanisms, application programmers with information about the impact of transaction management on performance, and system developers with ideas how to improve their systems with respect to transaction management.

In contrast to conventional transaction management mechanisms, the VODAK Transaction Manager utilizes the semantics of operations to increase parallelism of concurrent transactions. It is based on the principle of open nested transactions, implemented by nested two-phase locking. Different to conventional flat transaction mechanisms based on read/write operations on pages, VODAK transactions constitute a hierarchy of subtransactions dynamically constructed according to the invocation hierarchy of a particular method execution. Locks are requested by subtransactions and released as soon as the subtransaction commits or aborts, i.e. the method execution is finished. A commutativity relation on pairs of methods defines conflicts between subtransactions. This allows commuting methods of different transactions to be run in parallel even if they update shared data. For example, two add operations inserting integer values into a set that must not contain duplicates commute if they insert different values. Therefore, in VODAK they are allowed to run in parallel.

Recovery is handled by inverse methods that undo the effects of methods executed by a transaction that is to be aborted. Inverse methods and commutativity relations are defined in the VODAK Model Language (VML) together with the class definitions of the database schema. The example of Fig. 1 shows the definition of the above mentioned add method as part of a schema. The inverse method of add is remove. It removes a given integer value from the set. If an add operation did not insert its given value, because the value already existed in the set, remove must not be called in order to undo add. This is ensured by checking for the existence of a duplicate in the BEFORE clause, executed before the original add method, and by calling remove in the INVERSE clause only in case there was no duplicate. The COMMUTING clause defines commutativity between operations. In the example, all operations commute if they access different elements of the set.

```
add(o: INT): BOOL SUBTRANSACTION;
{
 VAR test: BOOL;
 test := (o IN contents);
 IF (NOT test)
 {
  INSERT o INTO contents;
  counter->incr(1);
 }
 RETURN test;
}
UNDO
 VAR contained: BOOL;
 BEFORE
  contained := (o IN contents);
 INVERSE
  IF (NOT contained) remove(o);

COMMUTING
 add(o): add(p), add(o): remove(p),  remove(o): remove(p)
 {
  RETURN (o != p);
 };
```

Fig. 1 : VML Schema Example

Fig. 2 shows the execution of the add method as an open nested transaction. add calls several system-defined methods, which are directly implemented as transactions of the VODAK storage manager. Incrementing the counter which reflects the number of elements contained in the set is executed as another subtransaction.

The demonstration uses a graphical user interface, providing for an interactive and dynamic definition of concurrent transactions and their execution, step by step. A screendump is shown in Fig. 3 . Intermediate transaction states as well as internal information about requested and already granted locks, blocked transactions, deadlocks and actions executed during rollback are displayed. Hence, the impact of executing different actions of concurrent transactions on the immediate states and internal structures can be inspected interactively. For each transaction executed during the demonstration, an independent decision can be made whether the transaction is executed as an open nested transaction or as a conventional flat transaction. Flat transactions keep their locks until the end of the

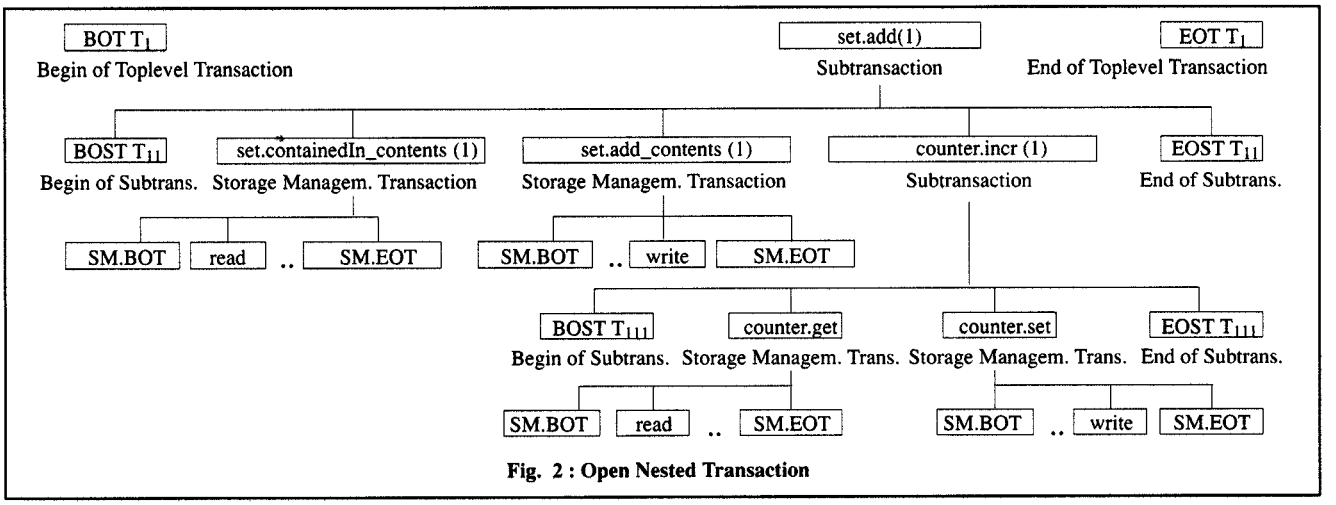

Fig. 2 : Open Nested Transaction

user-transaction. In the demonstration, one can easily notice the difference between both execution models in terms of probability of blocking and deadlocks, visualized by "tracing" internal information structures.

2. Background

VODAK has been designed to provide a powerful data model for storing and manipulating complex data like hypertext documents and knowledge bases. Currently, VODAK is extended to support multi-media applications. A prototype managing analog and digital video clips has been finished (see V^3 Video Server demonstration description in this volume). The question of efficiency and performance of multi-user access to object-oriented database systems has been addressed by the open nested transaction model. It has been implemented in VODAK. The conflict definition between methods in terms of commutativity relations and the definition of inverse methods for transaction undo is included in the corresponding VODAK database schema. For frequently used system defined methods the definition is part of the predefined VODAK kernel. Definitions for application defined methods are done by the schema developer.

In the near future, the VODAK transaction model will be extended to support the integration of heterogeneous autonomous database systems. A VODAK transaction can then spawn several heterogeneous database systems to provide VODAK users with homogeneous access to heterogeneous databases.

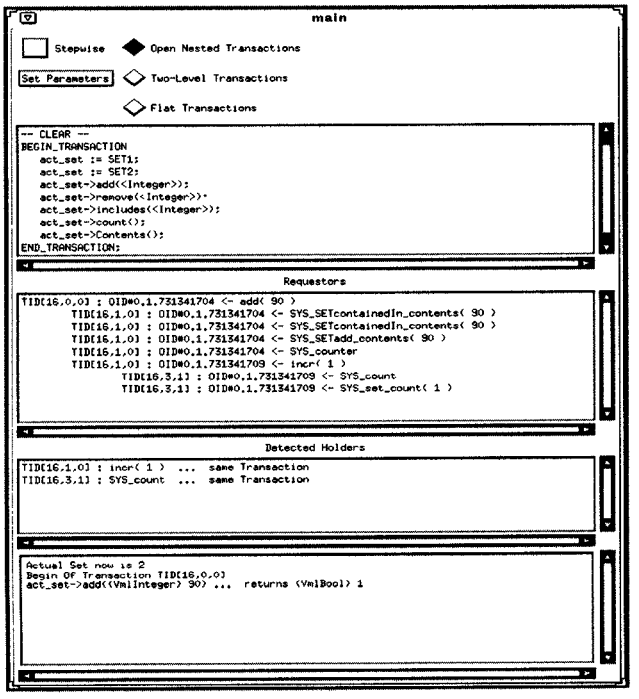

Fig. 3 : User Interface Screendump

3. Hard- and Software Platforms

The VODAK database management system is running on a SUN Sparcstation 2 or 10 with SunOS 4.1.x and OSF Motif. At least 20 MB shared memory, 20 semaphores and 100 MB of swap are required.

Acknowledgement: We are indebted to Achim Kraiß and Hannelore Eisner for their help in implementing this demonstration.

4. References

[1] **W. Klas et. al.**: "VML -VODAK Model Language V3.0", Specification Document, 1992, GMD-IPSI.

[2] **P. Muth, Thomas C. Rakow**: "Atomic Commitment for Integrated Database Systems", Proceedings of IEEE Seventh International Conference on Data Engineering, Kobe, Japan, April 8-12, 1991. Los Alamitos: IEEE 1991.

[3] **P. Muth, T. C. Rakow, W. Klas, E. J. Neuhold**: "A Transaction Model for an Open Publication Environment", in Ahmed K. Elmagarmid (Ed.): Database Transaction Models for Advanced Applications. Morgan Kaufmann Publishers, San Mateo, Ca., 1992.

[4] **P. Muth, T. C. Rakow, G. Weikum, P. Brössler, C. Hasse**: "Semantic Concurrency Control in Object-Oriented Database Systems", accepted for publication: Proceedings of IEEE Ninth International Conference on Data Engineering, Vienna, Austria, April 19-23, 1993. Los Alamitos: IEEE 1993

[5] **E. J. Neuhold, V. Turau (Eds.)**: "Database Research at IPSI", SIGMOD RECORD Vol. 21, No. 1, March 1992, pp.133-138.

[6] **T. C. Rakow, J. Gu, E. J. Neuhold**: "Serializability in Object-Oriented Database Systems", Proceedings of IEEE Sixth International Conference on Data Engineering, Los Angeles, Feb. 1990.

[7] **G. Weikum, C. Hasse, P. Broessler, P. Muth**: "Multi-Level Recovery", Proceedings of Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), Nashville, Tennessee April 2-4, 1990.