In: *Proc. of the 5th Workshop on Transaction Concepts*, Schloß Dhaun, Germany, Jan. 1994.

Capturing more Semantics with a Concurrency Control Technique Tailored to KBMSs

Fernando de Ferreira Rezende
Department of Computer Science - University of Kaiserslautern
P.O.Box 3049 - 67653 Kaiserslautern - Germany
e-mail: rezende@informatik.uni-kl.de - Phone: +49 (0631) 205 3274

Abstract

Knowledge Base Management Systems (KBMSs) are a growing research area finding applicability in many different domains. The higher its demand, the greater the necessity for knowledge sharing. In the near future, KBMSs will be applied more and more in real world applications [BM86]. As a matter of fact, the research for concurrency control techniques tailored to the KBMS environment plays a crucial role to this applicability. Moreover, it assumes a paramount importance as the demand for ever larger knowledge bases (KBs) grows. Trying to work out a solution to the problems appearing with parallel accesses in KBs, we propose a concurrency control technique tailored to KBMSs.

The literature on concurrency control algorithms is really vast. Among the most important classes of such algorithms are *locking*, *timestamps*, and *serialization graphs* [BHG87]. In particular, the class of locking-based algorithms has shown its practicality and performance. Additionally, locking-based algorithms have special solutions for graph structures, the abstraction for KBs that appears to be the most appealing [CHM92, Ch93]. Consequently, we have chosen to develop our concurrency control method for KBMSs based on locking. More specifically, we hope to be able to use the power and elegance of granular locks [GLPT76] in the KBMS environment. However, a protocol like the granular locks is designed for a single organization hierarchy, extended to directed acyclic graphs in case of index structures. As soon as we have many overlapping hierarchies, like the abstraction hierarchies, we may get into trouble with implicitly locked objects, because an implicit lock on a child object is only visible if it is accessed through a specific path of the graph. Our main idea then is to have implicit locks for objects in the subtrees, but at the same time to cope well with such problems.

As a matter of fact, a KB graph is built through the superposition of the classification/generalization, association, and aggregation hierarchies (or in fact graphs). However, many accesses in a KB are directed to a particular hierarchy, and not to the KB graph as a whole. Due to that, in our protocol we are going to logically partition the KB graph into those three main hierarchies. Hence, what we obtain is a combination of different abstraction hierarchies, and what we plan to do is to apply hierarchical lock schemes on each one of them. By such a way, on one hand we acquire a minimization of the locks in comparison with for example a simple approach, where every touched object must be locked. On the other hand we define more precisely the granule of lock to be accessed by a transaction, allowing it to lock just the objects it really needs to access.

Therefore, we provide users with the possibility of looking at a KB, and abstracting from it just the viewpoint to be worked out. In addition, such logical partitions are mirrored in each object of the KB. Hence, a user can abstract from the whole KB, just an object and its instances, or an object and its elements, and so forth. In order to capture more of the semantics contained in a KB graph and to provide a teeming utilization of the parallelism, we have created then three distinct sets of lock types, which are based on the logical partitioning of the graph previously introduced. Hence, similar to the granular locks protocol [GLPT76], we have a *basic set* of lock modes, named: IR (Intention Read), IW

(Intention Write), R (Read), RIW (Read Intention Write), and W (Write). However, we have this basic set of lock modes to each one of our logical partitions, i.e., to the classification (recognized by a subscript c ($_c$) following the lock mode), association ($_s$), and aggregation ($_a$) graphs, and not to the whole structure as in the granular locks. We named those locks as pertaining respectively to the sets of *C_type*, *S_type*, and *A_type locks* (in general, we call them *typed locks*). Therefore, the lock manager receives the lock requests, and by means of these typed locks, it can precisely identify which kind of descendants the requester is trying to reach. So, for example, if a user requests a W_c lock on an object, as soon as the lock is granted, the user has exclusive access to this object and implicitly to all its subclasses and instances. Additionally, our protocol provides also conventional read and write locks for transactions which want to access just an object as a closed unit, i.e., without worrying about the role of this object in the current state of the KB and its relationships. These locks are used as a complement to the typed locks.

In summary, our protocol has four main important characteristics. First, it offers different granules of locks, varying from a single object (handled by the conventional locks) to different sets of objects (typed locks). Second, it considers implicit locks, alleviating the task of managing too many locks due to the high number of objects in real world applications. Thirdly, it copes well with the several access paths to objects, by means of the requirement of explicitly locking objects with multiple parents, which in turn relaxes the necessity of covering all paths to the root with intentions, reducing it to only one path. Fourth, it interprets the relationships between objects with respect to their semantics, providing typed locks for all abstraction concepts. At last, such power, flexibility, and parallelism are by no means proffered by related works, as for example the one of [CHM93].

Finally, we capture more of the semantics contained in the KB graph in the sense that we do not consider descendants of an object as being merely meaningless descendants of it, but, on the contrary, descendants with special characteristics and significance, which are based on the abstraction relationships of generalization, classification, association, and aggregation. This is the most important point of our technique, by means of which we can really obtain a high degree of concurrency, with a full exploitation of all possible parallelism in a knowledge representation approach.

References

[BHG87] Bernstein, P.A., Hadzilacos, V., Goodman, N.: *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Publishing Company, MA, USA, 1987.

[BM86] Brodie, M., Mylopoulos, J. (Eds.).: *On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies*. Springer Verlag, 1986. (Topics in Information Systems).

[Ch93] Chaudhri, V.K.: *On the Performance of a Multi-User Knowledge Base Management System*. Internal Report, University of Toronto, Toronto, Canada, Dec. 1993. (submitted for publication).

[CHM92] Chaudhri, V.K., Hadzilacos, V., Mylopoulos, J.: Concurrency Control for Knowledge Bases. In: *Proc. of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, USA, 1992. pp. 762-773.

[GLPT76] Gray, J.N., Lorie, R.A., Putzolu, G.R., Traiger, I.L.: Granularity of Locks and Degrees of Consistency in a Shared Data Base. In: *Proc. of the IFIP Working Conference on Modeling in Data Base Management Systems*, Freudenstadt, Germany, Jan. 1976. pp. 365-394.