Concurrent Programming vs. Concurrency Control†

Shared Events or Shared Data

Bruce Martin††

Computer Systems Research Group
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, California 92093

Two views of concurrency in an object system exist. Those pursuing *concurrent programming* believe that *activities* in the real world are inherently concurrent and therefore objects are themselves active. Objects engage in shared events by sending and receiving messages. Communicating Sequential Processes [Hoar85a] and Actors [Agha86a] embrace this view. On the other hand, those pursuing models of *concurrency control* believe that objects are *data* and that concurrent access to data needs to be controlled by the system according to some correctness notion. Database transactions, atomic objects [Weih84a, Schw84a] and nested objects [Mart88a] embrace this view.

Concurrent programming, in our view, places a significant burden on programming. Correct concurrent behavior is specified as combinations of interactions within a potentially large set of concurrent objects. A programmer must verify that the implementations of all the objects never produce undesirable interactions. Correctness of concurrent behavior is left to the programmer.

We are pursuing models embracing concurrency control primarily because a programmer is not required to consider concurrency. The operations on an object can be specified in terms of preconditions and postconditions and traditional program verification techniques can be used to verify an operation's implementation. A programmer only considers the serial behavior of an object in *isolation*; he need not concern himself with how other concurrent activities might affect the object. Correctness of interleavings is left to the system.

Serializability is the usual correctness notion for concurrency control algorithms. In transaction terminology, each competing transaction executes a sequence of basic *actions*. Any interleaving of the actions is correct if it is equivalent to some serial execution of the transaction. Serializability allows a transaction to be programmed in isolation, that is without considering possible interleavings with other

†This work was sponsored in part by grants from U.C. MICRO Program and the NCR Corporation.
†† Authors current address: Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, California 94304
martin@hplabs.hp.com

transactions. The system may indeed interleave the actions of several transactions but it is up to the system to make the interleaving appear serial.

Concurrent programming is apparently more general. A programmer can implement anything, including undesirable interactions like deadlock. The price for this generality is that the programmer must reason about global orderings of events and thus correctness is difficult to show.

The traditional transaction model is not general enough for programming shared object systems. For example, several researchers, [Bern87a, Garc87a, Pu88a], have recognized that transactions are too restrictive for long-lived activities. The problem is that the transaction model is too conservative. Only reading and writing a data item at a single layer of abstraction is modeled. Once a read-write, write-read or write-write dependency is established between two transactions, it remains for the life of the transaction and limits further interleavings.

Our approach is to discover and explore less restrictive correctness notions that still allow programmers to implement operations on objects in isolation. In [Mart88a] we present two such correctness notions: *externally serializable computations* and *semantically verifiable nonserializable computations*. Both correctness notions assume the nested object model. In [Mart87a] we give a nested object solution to the Dining Philosophers' Problem [Dijk71a]. Nested objects incorporate both the *semantics* of an object and the data abstraction *hierarchy* of an object.

Nested objects form a nested object system. A nested object system is hierarchical; objects exist at different levels of the system. The execution of an operation on an object at level i results in the execution of operations on objects at level i-1. However, only top level objects are viewed externally.

A computation at level i is a description of the state change made to level i objects and the return values produced by executing a partially ordered set of operations on level i objects. The computations at each level together form an n-level system computation.

Externally serializable computations are n-level system computations in which the top level objects are left in states that could be produced by serial computations. However, lower level objects may be left in states that no serial computation could produce. Because both data abstraction hierarchies and operations semantics are considered in the nested object model, dependencies established between concurrent computations can be systematically ignored. Long-lived computations can execute efficiently if dependencies can later be ignored.

Nested objects are more general than other models of concurrency control. Transactions are two-level nested objects that read and write basic data items. Atomic objects are two-level nested objects that perform abstract operations.

The 1988 Object Based Concurrent Programming Workshop did not directly address the differences between concurrent programming and concurrency control. Perhaps future workshops can contrast the generality, the applicability, the programmability, the security and the performance implications of models

from both concurrent programming and concurrency control.

References

Agha86a.
G. Agha, "Actors," Ph.D. dissertation, MIT Press, Cambridge, Massachusetts, 1986.

Bern87a.
P. Bernstein, "Database System Support for Software Engineering," *Proceedings of the Ninth International Conference on Software Engineering omputing*, 1987.

Dijk71a.
E. Dijkstra, "Hierarchical Ordering of Sequential Processes," *Acta Informatica*, vol. 1, pp. 115-138, 1971.

Garc87a.
Hector Garcia-Molina and Kenneth Salem, "SAGAS," *Proceedings of the 1987 ACM SIGMOD Conference*, pp. 249-259, 1987.

Hoar85a.
C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, 1985.

Mart87a.
B.E. Martin, "Modeling Concurrent Activities with Nested Objects," *Proceedings of the Seventh International Conference on Distributed Computing Systems*, West Berlin, West Germany, September, 1987.

Mart88a.
B.E. Martin, "Concurrent Nested Object Computations," Ph.D. dissertation, UCSD Department of Computer Science and Engineering, June, 1988.

Pu88a.Calton Pu, Gail E. Kaiser, and Norman Hutchinson, "Split-Transactions for Open-Ended Activities," *Proceedings of the Fourteenth International Conference on Very Large Databases*, 1988.

Schw84a.
Peter M. Schwarz, "Transactions on Typed Objects," Phd Thesis , pp. 1-159, Department of Computer Science, Carnegie-Mellon University, December, 1984.

Weih84a.
W.E. Weihl, "Specification and Implementation of Atomic Data Types," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, Massachusetts, March, 1984.