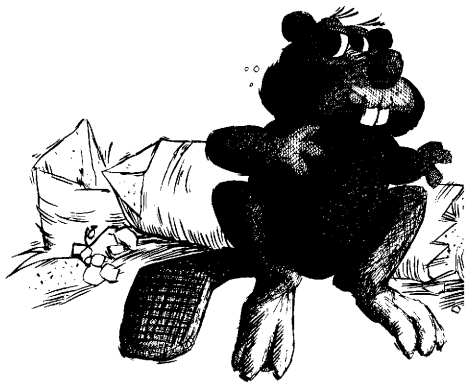
UNIVERSITY OF WATERLOO
UNIVERSITY OF WATERLOO
UNIVERSITY OF WATERLOO
COMPUTER SCIENCE DEPARTMENT
COMPUTER SCIENCE DEPARTMENT
COMPUTER SCIENCE DEPARTMENT



# *How Big Can an Atomic Action Be?*

*D.J. Taylor*

*CS-85-20*

*July, 1985*

# How Big Can an Atomic Action Be?

*David J. Taylor*

Department of Computer Science*
University of Waterloo
Waterloo, Ontario, Canada

*ABSTRACT*

It is sometimes implicitly or explicitly assumed that one should always be able to compose atomic actions to create larger atomic actions. This paper describes the properties normally required of an atomic action and examines them in the context of large (long execution time) actions. This leads to the conclusion that such large actions cannot reasonably be atomic.

## 1. Introduction

Atomic actions are now generally accepted as a fundamental principle for the control of concurrency in both centralised and distributed systems. In some cases, only one level of atomic action is considered, but usually the nesting of atomic actions is allowed. In particular, it is frequently asserted that when an atomic action is examined, it should be found to consist internally of a collection of smaller atomic actions, which themselves consist of yet smaller atomic actions, and so on, *ad infinitum*. The stopping point in this progression is taken to be arbitrary, depending only on viewpoint and convenience.

In some cases, it is assumed that this progression should also exist in the opposite direction. That is, given an atomic action, one should be able to identify other related atomic actions which can be packaged to form a larger atomic action, and so on. In this direction, the progression presumably stops when one reaches an atomic action which includes the entire universe, both in space and time.

The next section briefly summarises the properties usually demanded of an atomic action, then the following section discusses whether these properties can reasonably be expected of arbitrarily large actions. The last section then draws some conclusions.

---

*This work was performed while the author was a Visiting Research Scientist at the Computing Laboratory, University of Newcastle upon Tyne.

## 2. Properties of Atomic Actions

Many authors have discussed atomic actions (sometimes using the name "transaction"), using related but slightly different criteria to define an atomic action. This section is an attempt to describe briefly the various criteria used in a number of papers.

In a recent paper [9], Haerder and Reuter identified four criteria defining an atomic action ("transaction" in their terminology). They are: (1) *atomicity,* the action has its complete, desired effect, or no effect at all, (2) *consistency,* the action leaves a consistent system state if the system state was initially consistent, (3) *isolation,* there are no interactions with other atomic actions, and (4) *durability,* once an atomic action has completed, all changes it has made to the system state become permanent. The last property is modified slightly for nested actions: in that case, the permanence is only with respect to the enclosing action–if that action is aborted, the changes disappear. The consistency property may also be modified for nested actions, since different degrees of consistency may be required at different levels of nesting.

The four criteria specified by Haerder and Reuter provide a convenient basis for examining other authors' atomic action definitions. Although the Haerder and Reuter definition is not to be considered more authoritative, it is used as a standard of reference in the following discussion.

The very early paper by Eswaran [4] requires that each atomic action see a consistent system state and preserve that consistency. It also requires that concurrent excution of actions produce the same effect as serial execution in some order. The first requirement is essentially equivalent to the consistency and isolation criteria stated above. The latter requirement, generally known as *serialisability,* can be achieved by ensuring the isolation and atomicity criteria, if it is understood that atomicity implies the ability to undo a partially completed action. However, serialisability does not necessarily imply isolation and atomicity: if the implementation mechanism has knowledge of operation semantics, other techniques can be used to achieve serialisability.

Lomet [12] states three characterisations of an atomic action. (1) An atomic action does not observe any state changes by other actions, during its execution (the state appears static), and its state changes do not become visible until the action completes. (2) There is no communication between concurrent atomic actions. (3) An atomic action appears to be indivisible and instantaneous, hence concurrent execution of a set of actions is equivalent to some serial execution sequence. The first two of these are essentially the isolation criterion, but the third implies serialisability. The relationship between serialisability and the four criteria given initially has been discussed above.

Gray [8] requires atomicity, consistency, and durability of an atomic action ("transaction"), but does not mention isolation. Davies [3] simply requires that an atomic action perform all or none of its desired function, which is the atomicity criterion, although a property equivalent to isolation is also discussed in the paper. Liskov [11] makes two requirements: *failure atomicity,* which is equivalent to the atomicity criterion, and *indivisibility,* which is equivalent to the isolation criterion. A property called *permanence of effect* is also described, equivalent to the durability criterion. Liskov and Scheiffler [10] require two

properties of an atomic action: *indivisibility*, equivalent to the isolation criterion, and *recoverability*, equivalent to the atomicity criterion. Mueller, Moore, and Popek [13] require that an atomic action ("transaction") proceed *indivisibly*. This requirement is equivalent to the atomicity and isolation criteria.

Anderson and Lee [1] simply require that there be no interactions between concurrent actions, equivalent to the isolation criterion. Shrivastava [14] requires serialisability, which, as discussed above, can be achieved if both the atomicity and isolation criteria are met, but does not imply these criteria. Best and Randell [2] provide a formal definition of atomicity which cannot readily be reduced to a few words. However, their definition is essentially based on serialisability.

Thus, in terms of the requirements stated by Haerder and Reuter, most definitions of atomic actions require isolation, many also require atomicity. The requirements of consistency and durability, although often not explicitly stated, are also generally assumed. In some cases, a different approach is taken, and atomic actions are defined in terms of serialisabilty. This property is related to the others, in that atomicity and isolation together imply serialisability. The definition used by Gray [8] is an apparent anomaly, requiring neither isolation nor serialisability.

## 3. Large "Atomic Actions"

The paper by Shrivastava [14] is explicitly concerned with building large atomic actions, whose duration could be several days, or even several months. Gray [8] also explicitly considers atomic actions ("transactions") which could last a very long time. The same idea seems to be implicit in some other papers, which essentially assume that atomic actions can be used as the sole tool for structuring activities on a database or other system. (Although Haerder and Reuter [9] do state that "Typically, a transaction is a short sequence..." they do not make any stronger statement about transaction length.) Using atomic actions in a distributed system also tends to produce long running actions because of communication delays, particularly if many nodes are involved.

A fundamental question then is: "Can an atomic action reasonably last for an extremely long time?" It is clear that various implementation difficulties need to be resolved, and the paper by Shrivastava presents one approach to these difficulties. However, it does not consider whether the very long-lived entities under consideration are really atomic actions at all. The paper by Gray presents a different approach to these implementation difficulties. It does mention the possibility that these long-lived entities may be different from "ordinary" atomic actions, but dismisses the possibility.

In the remainder of this section, first the requirements listed by Haerder and Reuter, and used by many other authors, will be considered, then the serialisability criterion. Finally, the examples used by Shrivastava and Gray will be considered in the light of the general discussion.

The consistency requirement for atomic actions seems a desirable property for almost any activity on a database. Clearly it should be required of an activity regardless of its duration, although different degrees of consistency may be

specified at different levels of nesting. Thus, this one property presents no real problems.

The durability requirement also appears straightforward, since again it is desirable when an activity completes, no matter how long the activity, for the changes it has made to become permanent. There is a difficulty, however, when the nesting of atomic actions is considered. Durability is then only with respect to the enclosing atomic action, thus if an atomic action lasts for a week, the results of any internal atomic actions are not permanent in the ordinary sense of the word until the end of the week. Thus, unless some guarantee can be made that long-lived actions are not rolled back, very large amounts of work can be lost. The worst aspect of this is that external interactions which took place long in the past may be undone. For example, a data entry clerk will have to hope that an atomic action abort on Friday afternoon doesn't require him or her to redo some data entry performed on Monday morning. There seems no reason to believe that long-running actions will be any less prone to problems such as deadlock (indeed deadlocks seem much more likely), and hence no reason to believe that they will not sometimes have to be rolled back. Of course, the problem can be circumvented by eliminating such rollbacks, but this will certainly prevent achievement of the atomicity property, and probably the consistency and isolation properties as well.

It likely is possible to achieve the atomicity property for long-running actions, but at a very high cost. Systems which implement atomic actions usually abort any actions which are running when a system crash occurs (in a distributed system, when any node failure or network partition affects the atomic action). A long-running action which was treated in this way would probably never complete. Hence, it would become necessary to preserve at least some actions across system crashes and similar unfortunate events. This would require that information normally kept in volatile storage be kept in stable storage and would also greatly complicate the restarting of a crashed system. The impact of network partitions in such a situation requires further analysis, but a very serious impact on both sections of a partitioned network seems likely.

The isolation property seems untenable for a long-running action. Data cannot be kept "locked up" inside an action for arbitrarily long periods. In addition to the greatly increased probability of deadlock, there is the very basic requirement of obtaining reasonably current information from the system. A read-only action can always be made to work promptly by allowing it to access the state which existed prior to the start of all currently executing actions, but if this state is weeks old it is of very little value. The paper by Shrivastava attempts to avoid this problem by allowing actions to release tentative output values before they terminate. However, if these values are used by, for example, a report generator, we are faced with the prospect of seeing almost every element in the output report followed by "(this may or may not be correct)."

Finally, there is the serialisability criterion. Based on the above, it cannot be achieved in the usual way, by enforcing the atomicity and isolation criteria. However, it is also possible to consider serialisability independent of any particular implementation. Suppose there is a set of atomic actions each of which lasts on the order of weeks. Suppose there is also an atomic action which

reports how many widgets have been sold. If the long-running actions have anything to do with widget sales, then it is important to establish some serial order for the reporting action and the other actions. If the reporting action logically precedes the currently executing actions, it can be executed immediately but it will give a sales figure which is weeks out of date. If it logically follows the currently executing actions, it can't be executed until they have terminated, in which case we don't find out for weeks how many widgets have been sold, and the figure we obtain is still out of date when we finally receive it. Using the approach in the Shrivastava paper, the reporting action can be run immediately, using uncommitted results, and will produce an output such as "2756 widgets have been sold (maybe)."

Having discussed the application of the criteria for atomic actions to long-lived actions, it is useful to consider some specific examples. Since the Shrivastava and Gray papers explicitly discuss such actions, it is appropriate to use the examples from those papers.

The first example used by Shrivastava is not fully developed, but concerns the processing of an insurance claim. The assertion is that such an action might last six months. Presumably, the action includes such lower-level actions as receive report of loss, receive report of claims adjuster, evaluate amount to be paid, issue cheque. The sequence is likely a number of data recording actions followed by a few actions which complete processing of the claim. If the entire action really is an atomic action, then the clerk who performed the initial data entry action could be asked to perform it again up to six months later. As well, trying to execute an action which determines how productive claims adjusters have been recently will suffer from the same problem as occurred above in determining widget sales. Thus, this six-month long activity cannot reasonably be an atomic action.

The second example is a distributed calendar system, allowing, among other things, the scheduling of meetings among participants in the calendar system. The assertion is that the entire process of scheduling a meeting, from tentative location of possible time slots through to final confirmation should be a single atomic action. However, it is clear that several such ongoing actions must interact with each other if the calendar system is to be of any use. Thus, neither isolation nor serialisability can be achieved. The example, using techniques of the paper, does achieve a useful calendar system, but the "atomic actions" are not serialisable and thus do not meet the definition of atomic action presented in the paper.

Gray presents as an example the making of an extensive set of travel arrangements, involving airlines, hotels, etc. It is claimed that the entire sequence of activities related to a single trip should be considered an atomic action. It is again clear that neither serialisability nor isolation can be achieved. Interestingly, neither of these is required by the definitions of that paper, so the example is reasonable in terms of the paper containing it. However, as observed previously, the omission of both serialisability and isolation distinguishes that paper from all the others cited. In practice, it seems certain that one would want these properties to hold for the shorter-lived actions embedded in the long-lived action.

## 4. Conclusions

The material presented in the preceding section indicates that very long-lasting activities cannot reasonably be treated as atomic actions. It is not possible to provide a precise threshhold, but likely any action lasting for even several minutes should be viewed with suspicion. It is not claimed that atomic actions are unsuitable for distributed systems, but caution is required because distribution will tend to increase execution time. Thus, while it is appealing to use a single principle, such as atomic actions, regardless of scale, in this case it appears not to be possible. In particular, only Gray's definition is suitable for large actions, and it is unsuitable for small actions.

To some extent, the inability to build very large atomic actions simply reflects the non-atomicity of long-lasting events in the real world. Garcia-Molina [6] discusses an example involving transfer of funds within a banking system. In such examples, it is generally assumed that preventing any loss of funds is an essential requirement, but he states: "Furthermore, money is lost in the real world anyway, so why shield the database users from this fact?" Even when an activity is short enough that it seems intuitively appealing to treat it as an atomic action, practical considerations may make it impossible. In particular, since the probability of deadlock is dependent on the number of atomic actions in the system and the number of resources each holds, a very busy system may be able to tolerate only very short atomic actions. An airline reservation system is an example of such a very busy system: there, the process of booking a seat is broken into a number of atomic actions, but the whole process is not atomic [7]. Essentially, in such systems, an engineering decision has been made to limit atomic actions so that no interaction with the environment, including the individual invoking the action, takes place during the action. Thus, the duration of each action can normally be limited to a fraction of a second.

Clearly, there are many cases, such as the above example, in which sets of related atomic actions extend over (relatively) long intervals. A method of structuring them should be sought, but another level of atomic action will not do, so some alternative approach is required. (I am tempted to call this alternative a "molecular action," but I resist the temptation.) Such actions will have weaker constraints than atomic actions, most notably as regards isolation. Likely, the central difficulty is in formulating a criterion which can be used to determine whether an execution sequence of such actions produces an acceptable result. In the case of atomic actions, equivalence to a serial execution sequence provides a straightforward test, but, as argued above, such a requirement is inappropriate here.

Fleisch [5] has discussed the idea of grouping atomic actions into larger units called *jobs* with the requirement that a job be *coherent*. Unfortunately, the concept of coherence is not given a precise, formal definition, so it is difficult to evaluate the general applicability of the concept. Another distributed system which has addressed the problem, from a practical standpoint, is TABS [15]. In TABS, four properties for actions are defined, but a programmer may choose to implement an action which does not possess one of the standard properties. This provides an efficiency/complexity tradeoff for the programmer. It does not

appear that a weaker version of any of these properties is available: it is simply a binary choice for each property.

Researchers have been investigating such long-lived actions, but under the assumption that the objects under investigation are essentially atomic actions, with minor modifications. Here it has been argued that they are fundamentally different. It may be that better approaches to the problems of long-lived actions can be obtained by recognising the fundamental difference. Most importantly, one should not try to find a single, universal solution. Rather, one solution is required for atomic actions, and a second solution for the long-lived actions containing atomic actions.

**References**


1. T. Anderson and P. A. Lee, *Fault Tolerance: Principles and Practice*, Prentice-Hall, Englewood Cliffs, N.J. (1981).
2. E. Best and B. Randell, A formal model of atomicity in asynchronous systems, *Acta Informatica* **16**(1) pp. 93-124 (1981).
3. C. T. Davies, Data processing spheres of control, *IBM Systems Journal* **17**(2) pp. 179-198 (1978).
4. K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger, On the notions of consistency and predicate locks in a data base system, *Communications of the ACM* **19**(11) pp. 624-633 (November 1976).
5. B. D. Fleisch, META-Activities: Towards coherent distributed jobs, *Proceedings, The 4th International Conference on Distributed Computing Systems*, pp. 566-578 (May 14-18, 1984).
6. H. Garcia-Molina, Using semantic knowledge for transaction processing in a distributed database, *ACM Transactions on Database Systems* **8**(2) pp. 186-213 (June 1983).
7. D. Gifford and A. Spector, The TWA reservation system, *Communications of the ACM* **27**(7) pp. 650-665 (July 1984).
8. J. N. Gray, The transaction concept: Virtues and limitations, *Proceedings, Seventh International Conference on Very Large Data Bases*, pp. 144-154 (September 9-11, 1981).
9. T. Haerder and A. Reuter, Principles of transaction-oriented database recovery, *Computing Surveys* **15**(4) pp. 287-317 (December 1983).
10. B. Liskov and R. Scheifler, Guardians and actions: Linguistic support for robust, distributed programs, *ACM Transactions on Programming Languages and Systems* **5**(3) pp. 381-404 (July 1983).
11. B. Liskov, On linguistic support for distributed programs, *IEEE Transactions on Software Engineering* **8**(3) pp. 203-210 (May 1982).
12. D. B. Lomet, Process synchronization, communication and recovery using atomic actions, *SIGPLAN Notices* **12**(3) pp. 128-137 (March 1977).

13. E. T. Mueller, J. D. Moore, and G. J. Popek, A nested transaction mechanism for LOCUS, *Proceedings of the Ninth ACM Symposium on Operating System Principles*, pp. 71-89 (October 10-13, 1983).

14. S. K. Shrivastava, A dependency, commitment and recovery model for atomic actions, *Proceedings, Second Symposium on Reliability in Distributed Software and Database systems*, pp. 112-119 (July 1982).

15. A. Z. Spector, et al, Support for distributed transactions in the TABS prototype, *Proceedings, Fourth Symposium on Reliability in Distributed Software and Database Systems*, pp. 186-206 (October 15-17, 1984).