Progress in the Garbage Collection of Active Objects

Dennis Kafura
Department of Computer Science
Virginia Polytechnic Institute
and State University
Blacksburg, VA 24061
email: kafura@vtopus.cs.vt.edu
phone: 703-231-5568

Doug Washabaugh
Digital Equipment Corporation
tay2-2/b4
153 Taylor Street
Littleton, MA 01460
email:washabaugh@quiver.enet.dec.com
phone: 508-952-3535

Jeff Nelson
Digital Equipment Corporation
ZK02-3/N30
110 Spitbrook Road
Nashua, NH 03062
email: jnelson@tle.enet.dec.com
phone: 603-881-0867

Introduction

Research in object based concurrency has exposed a novel and difficult garbage collection problem. The novelty and the difficulty of the problem arise for the same reason - the unit of reclamation is an entity that possesses a thread of control. Such objects, termed *active* objects or *concurrent* objects, are explored in our work through the actor model of concurrency [Agha 1986].

In this paper we summarize our work to date on the garbage collection of actors by making the following points:

- motivation: active objects are useful and they should be managed by automatic (i.e., garbage collection) techniques rather than by programmer-developed code.
- uniqueness: the garbage collection of actors, and of active objects generally, is fundamentally different from other garbage collection problems.
- status: garbage collection of actors can be done incrementally and concurrently in either, but not both of, distributed systems or real-time systems.

We conclude this overview by noting some important areas of current and future work.

Motivation

Objects which encapsulate a thread of control as well as code and data are useful because:

- parallelism inherent in a system of concurrent objects can be exploited readily by parallel architectures [Athas 1987],

- asynchrony among entities in the real world can be represented directly by concurrent objects [Kafura 1988],

- distributed applications can be programmed "naturally" using concurrent objects interacting via messages [Black 1987] [Yonezawa 1987],

- conceptual economy results from a single object abstraction which unifies the notions of a processor (thread of control), memory (encapsulated variables) and communication (messages).

Automatic methods (i.e., garbage collection) for managing memory are preferable because:

- a better division of responsibility results when the system does what it does best (manage resources) and the programmer does what programmers do best (design systems),

- since programmer control of passive (data) objects is notoriously error-prone [Bloom 1987], we believe that it will be even more difficult for a programmer to correctly manage concurrent objects because of the added complexity of concurrency,

- the penalty of programmer errors in managing active objects is more pronounced because garbage concurrent objects consume not only memory space but may also consume processing capacity.

- in distributed or real-time applications it is unlikely that, or at least excessively expensive for, a programmer to devise a correct and efficient algorithm for managing active objects in distributed or real-time applications.

Despite these motivations little work has been done on this problem. Notable exceptions are related works in functional programming [Hudak 1982] [Hudak 1983], the Emerald system [Jul 1988] and message passing models [Baker 1977] [Halstead 1978].

Uniqueness

The criterion pervasively used in collectors for passive (data) objects is reachability. To see that this criterion is insufficient for collecting active objects, consider the actor system shown in Figure 1. In this figure, actors A and I are root actors and by definition are not garbage. It can be easily seen that actors J,K,L and M are garbage - they cannot communicate with a root actor. Whatever actions they take cannot be made visible to the outside world. Notice that J and M are active while K and L are blocked. This shows that state alone is not a sufficient criterion for identifying garbage active objects. Actor E is blocked and there is no way for it to become active because it has no inverse acquaintances. However, actor H also has no inverse acquaintances but it is not garbage because it is active and can communicate directly with the root actor I. Message from the root actor A can reach actors B,C,D and G. If these messages contain A's mail queue address, these four actor can become active and communicate directly with a root actor. Hence, they are not garbage. Finally, actor F could send a message to G containing F's own mail queue address. G in turn could send A's mail queue address to F allowing F to communicate with the root actor A. So F is not garbage.

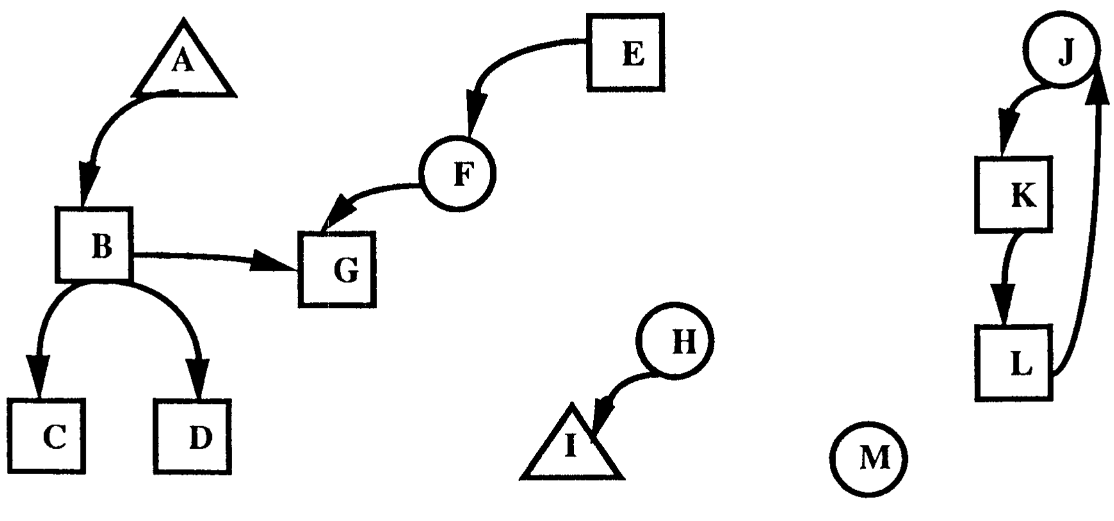

Figure 1. An Actor System

Notice that for the system shown in Figure 1, actors E,F and H are not reachable from a root. If a traditional marking algorithm is used, these three actors would be incorrectly marked as garbage. Also note that reference counting can error in two ways. First it can miss actors which are garbage such as actors J, K and L all of which have non-zero reference counts even though all of them are garage. Second, actor H is not garbage even though it has a zero reference count.

Status

The collector we have developed is based on a set of coloring rules [Nelson 1989] implemented by a co-routine algorithm called the Push-Pull algorithms [Kafura 1990b]. This algorithm and its extensions allow for the following goals to be achieved.

<u>concurrent mutator/collector</u>

To allow for concurrent execution, the mutator and collector must cooperate in two ways. First, they must synchronize their access to shared implementation structures. This is straightforward. What is harder is the second form of cooperation: the collector must take a snapshot of the actor system on which it will work while allowing the mutator to migrate away from this snapshot state.

<u>incremental collection</u>

When the mutator and collector are executed on a single processor system, concurrent mutator/collector operation may imply that the mutator is interrupted for long periods of time. It is useful to minimize the length of this interruption by interleaving incremental actions of the collector into allocation operations of the mutator. This is not too difficult with the Push-Pull algorithm because a "few" steps in the algorithm can be performed at each allocation. [Washabaugh 1990a].

real-time collection

Not only should the collector work incrementally, but the period of time during which the mutator is interrupted must be strictly bounded. Furthermore, it must still be guaranteed that the entire reclamation process is completed before the mutator consumes all available memory. The incremental extension of the Push-Pull algorithm can be strictly bounded [Washabaugh 1990b].

distributed collection

Collecting distributed garbage presents two major problems. First, the global collector must operate concurrently with the local collectors/mutators and must synchronize properly with the local collector. This synchronization can be achieved again by using a snapshot approach and by "time-stamping" inter-node acquaintances. Second, the distributed pieces of the global collector must be able to determine when to terminate. The termination is complicated because a global collector at one node may finish all of its work only to be reawakened later by the action taken at another node. Agreement can be achieved by using a rotating token which, if it ever returns to its last "owner", signals termination. [Washabaugh 1990a]

Future Work

There are several limitations of the collectors we have designed to date.The work needed to remove these limitations, are:

- study the performance of the garbage collector by empirical observation or by simulation experiments. Such studies may lead to more efficient algorithms as the behavior of the collector and the patterns of garbage actors are better understood. These studies are particularly important for real-time applications.
- instill fault tolerance into the distributed collector.
- investigate compilation techniques which might reduce the garbage collector's load. For example, it is possible in certain cases to merge actors.
- extend the garbage collector for variations of the actor model (e.g., ACT++ [Kafura 1990a]).
- formally prove the correctness of the collection algorithms. Thus far only informal arguments and testing have been used.

These research items are currently under investigation.

References

[Agha 1986] Gul Agha, Actors: A Model of Concurrent Computation in Distributed Systems, M.I.T. Press, Cambridge, Massachusetts, 1986.

[Athas 1987] W. Athas, "Fine Grain Concurrent Computations," Technical Report 5242:TR:87, Computer Science Department, California Institute of Technology, 1987.

[Baker 1977] Henry Baker and Carl Hewitt, "The Incremental Garbage Collection of Processes," M.I.T. Artificial Intelligence Laboratory, Memo 454, December 1977.

[Black 1987] Andrew Black, Norman Hutinson, Eric Jul, Henry Levy and Larry Carter, "Distribution and Abstract Types in Emerald," IEEE Transactions on Software Engineering, Vol. SE-13, No. 1, January 1987, p.65-76.

[Bloom 1987] Tony Bloom and Stanley Zdonick, "Issues in the Design of an Object-Oriented Database Programming Language," OOPSLA'87, October 1987, p.441-451.

[Halstead 1978] Robert Halstead, "Multiple-Processor Implementations of Message Passing Systems," M.I.T. Laboratory for Computer Science, Technical Report 198, April 1978.

[Hudak 82] Paul Hudak and Robert Keller, "Garbage Collection and Task Deletion in Distributed Applicative Processing Systems," Symposium on Lisp and Funtional Programming, 1982, p.168-178.

[Hudak 1983] Paul Hudak, "Distributed Task and Memory Management," 2nd Annual ACM Symposium on Principles of Distributed Computing, 1983, p.277-289.

[Jul 1988] Eric Jul, Henry Levy, Norman Hutchinson and Andrew Black, "Fine-Grain Mobility in the Emerald System," ACM Transactions on Computer Systems, Vol. 6, No. 1, February 1988, p.109-133.

[Kafura 1988] Dennis Kafura, "Concurrent Object Oriented Real-Time Systems Research," Technical Report 88-47, Department of Computer Science, Virginia Tech, Blacksburg, VA, 1988.

[Kafura 1990a] Dennis Kafura and Keung Lee, "ACT++: Building A Concurrent C++ With Actors," Journal of Object-Oriented Programming, May, 1990.

[Kafura 1990b] Dennis Kafura, Doug Washabaugh and Jeff Nelson, "Garbage Collection of Actors," to appear in: Proceedings 1990 ECOOP/OOPSLA Conference, October, 1990.

[Nelson 1989] Jeff Nelson, Automatic, Incremental, On-the-fly Garbage Collection of Actors, M.S. Thesis,Department of Computer Science, Virginia Tech, Blacksburg, VA, February 1989.

[Washabaugh 1990a] Doug Washabaugh, Real-Time Garbage Collection of Actors in a Distributed System, M.S. Thesis, Department of Computer Science, Virginia Tech, Blacksburg, VA, February 1990.

[Washabaugh 1990b] Doug Washabaugh and Dennis Kafura, "Incremental Garbage Collection of Actors for Real-Time Systems," to appear in: Proceedings of the 11th Real-Time Systems Symposium, December 1990.

[Yonezawa 1987] A. Yonezawa, E. Shibayama, T. Takada, and Y. Honda, "Modelling and Programming in an Object-Oriented Concurrent Language, ABCL/1," in Object-Oriented Concurrent Programming (A. Yonezawa and M. Tokoro, eds), p.55-89, MIT Press, Cambridge, Massachusetts, 1987.