Position Statement on Concurrent Objects for Massively Parallel Architectures*

Edward A. Luke / Electrical Engineering
Helen C. Takacs / Computer Science
hctakacs@msstate
William C. Welch / Institute for Technology Development
claude@cs.cmu.edu
Mississippi State University

1. Introduction

In choosing the appropriate model for programming fine-grained tasks to be run on a multicomputer system the facts that objects are inherently sequential and actors are inherently concurrent do not strongly influence the choice. The interesting concurrency occurs in the interaction among computational agents, not in the concurrency within an agent.

2. Motivation

This discussion reviews some features of objects and actors and attempts to answer remarks made by Carl Hewitt [Hewitt, 1988} concerning the superiority of actors over objects.

Our interest in this discussion stems from research on a multicomputer being designed at Mississippi State University called the Mapped Array Differential Equation Machine (MADEM). The architecture for this floating-point-intensive scientific problem solver supports a fine-grained reactive message-passing programming model. Message delivery and process scheduling are performed by a node's i/o processor, which operates concurrently with a high performance numeric processor. In choosing our model for concurrent computation we were influenced by research at Cal Tech and by the work of William Dally [Dally, 1986].

3. Concurrency within computational agents

It has been said [Hewitt, 1988] that actors are not objects, because objects are inherently sequential, while actors are inherently concurrent. The essential concurrency in an object-based model occurs because multiple asynchronous threads of activity use message passing to share data and to synchronize their actions. Within these threads of activity, execution is sequential. Indeed, objects can be viewed as combining sequential procedures and declarative information to control access to data and to promote modularity. In our system "objects" and "processes" are interchangeable except that the object notation affords encapsulation and information hiding. On the other hand, an actor has no presumed sequentiality in its actions.

Clearly actors support a finer grain of concurrency. After receiving a message, an object performs a sequence of actions (which may include instancing other objects), and then either terminates or grabs the next message in its queue. An actor responds to a message by executing a

*The research described in this paper was sponsored by the Defense Advanced Research Projects Agency under contract number DAAA 15-86-K-0025, through a subcontract with Mississippi State University set of actions, too, but these may be performed in any order or in parallel, including the computing of the actor's "replacement behavior" (i.e. specifying the actor that will process the next message in the queue.) This notion of allowing processing on the next message in the queue to begin before processing on the current message has completed is difficult to mimic with concurrent object systems, because a new object does not assume ownership of an old object's queue. The older object would have to relay messages from its queue to the newer object's queue explicitly. The message queue in the actor system is distributed, so that a replacement actor may start on a processor different from the one on which the current message is executing.

4. Lack of significance of intra-agent concurrency

If we grant that finer-grained tasks are essential to take advantage of massively parallel ensemble machines, then whether we view our individual computational agents as objects or as actors, the sequential nature of the objects or the parallel nature of the actors has little impact on the overall parallelism in the system. An object consists of a code area and a private memory area. The private memory area of an object in a fine-grained message-driven program is quite small, say less than 100 bytes. Each object is an independent computational agent, interacting with others solely by message passing. It consists of private variables that persist between receiving messages, a list of variables that describe the contents of the next message, and a sequence of actions describing how the object will react to the next message. There will be only a small number of instructions executed between each communication operation. The size of these threads will be so small that the decrease in concurrency caused by ordering actions in an object versus performing these actions in parallel is irrelevant to overall concurrency. As the grain-size decreases, less work is done in response to receipt of a message. We begin to view the actions taken in processing a message as an atomic response.

5. Summary

Replacement behavior in actors offers a level of concurrency that objects do not have, but as long as there is enough work to keep the processors busy in a parallel machine no additional concurrency is necessary. Handing off the replacement behavior to another processor does not accomplish anything if the processors have plenty to do already. Therefore, even though replacement behavior in the actor occurs anywhere, even perhaps before the method is performed, while replacement behavior in an object is always at the end, there is no advantage to early replacement behavior if the method is fine-grained. It is not as important to have "parallel" replacement behavior when it demands more from the communication network.

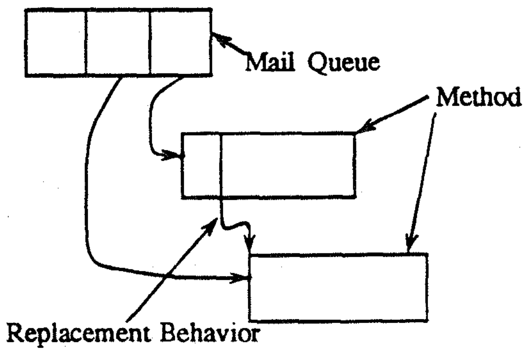

Figure 1.a:
In the actor model the replacement behavior occurs at any point in the method.

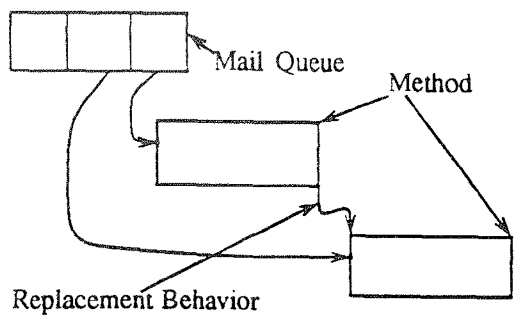

Figure 1.b:
In the object model the replacement behavior occurs at the end of an ordered method. In finer grained applications where the methods become more atomic in nature, this limitation is less significant.

The challenge is to write programs organized into small pieces that send and receive messages. We choose the model that facilitates programming. Researchers in parallel processing developed the actors model, while researchers in software engineering developed the object oriented model. It is not surprising then that actors can describe an additional level of concurrency, but objects facilitate organization of programs. Many new languages are borrowing from both models, *e.g.* [Athas, 1988]. Once the program is expressed as a collection of fine-grained objects, exploitation of concurrency is straightforward.

Efficient use of fine-grain machines, i.e. machines that contain tens of kilobytes of memory as opposed to those which contain megabytes of memory [Athas, 1988] remains on the research frontier. The interesting concurrency occurs in the interaction among computational agents, be they actors or objects. If message communication takes place in larger units and less frequently than memory accesses, then message communication between computational units can exhibit a larger latency. But as the sequences that execute between messages are reduced, then message latencies must reduce in proportion, for performance is end-to-end latency time for interprocess communication [Hewitt, 1988].

As message-passing performance improves relative to computing performance, the contrast between actors and objects may become significant, but present day medium-grain multicomputers cannot exploit the full advantages offered by the inherent concurrency of actors. We can be optimistic about the practicality of actors in our future.

References

[Agha, 1986]
Agha, G. A., *Actors: a model of concurrent computation in destributed systems*, Boston : MIT Press, 1986

[Athas, 1988]
Athas, William C. and Seitz, Charles L., Multicomputers : message-passing concurrent computers, *IEEE Computer*, 21, 8. (Sept. 1988) pp. 9-24.

[Dally, 1986]
Dally, W. J., *A VLSI architecture for concurrent data structures*, Hingham, MA : Kluwer Publishers, 1986.

[Hewitt, 1988]
Hewitt, Carl. Panel discussion on object-oriented concurrency. OOPSLA '87. *Sigplan Notices*. 23, 5. (May 1988) pp. 119-127.