# Variably Asynchronous, Reflective Tasks in Smalltalk[1]

a position paper for the OOPSLA '91 Workshop on Reflection and OOP

Jeff McAffer
Object Technology International
jeff@oti.on.ca

## Abstract

This paper describes the use of computational reflection in distributed simulation and illustrates that the results can be applied to the more general problem of scheduling and synchronization in concurrent systems.

## Introduction

What form does computational reflection take in a concurrent system? What might be its purpose? Reflection in a concurrent system should endow tasks with the capability to reason about and modify their concurrent behaviour. The literature contains several models of reflection (e.g., 3Lisp [des Rivieres and Smith 84], ABCL/R [Watanabe and Yonezawa 88], and ObjVlisp [Cointe 87]). For our research we have use the ABCL/R model and applied it to ENVY™[2]/Actra, a real-time multiprocessor Smalltalk.

ABCL/R is a concurrent object-oriented language which supports the notion of meta-objects. For each object, A, there is a meta-object, ↑A which describes A, just as A describes some entity in the problem domain. Meta-objects are similar to Smalltalk classes in that they define both the structural and computational aspects of other entities. Unlike normal objects, both A and ↑A execute in parallel (i.e., ↑A can change the behaviour of A while A is executing) and there is a unique meta-object for each object. Because both A and ↑A are objects, there also exists an ↑↑A, the meta-object for ↑A. This infinite chain of meta-objects is similar to the infinite tower of interpreters found in 3Lisp. *Level shifting* is done automatically when a meta-object receives a message. Thus, reflective methods for an object are implemented in its meta-object.

In our terminology, an ABCL/R object is an *actor* and a meta-object is a *meta-actor*. In the actual implementation of ENVY/Actra, an actor is a Smalltalk object with additional support for multiprocessing and communications. The ENVY/Actra virtual machine provides an interface to the messaging mechanisms of the underlying multiprocessing OS. Actors on a particular processor are scheduled by the local Smalltalk

---

[1] This research was supported in part by the Defence Research Establishment Ottawa and the Natural Sciences and Engineering Council of Canada.
[2] ENVY is a registered trademark of Object Technology International.

ProcessScheduler. The protocol used for message passing is based on the send, receive, reply primitive set found in the Harmony RTOS [Gentleman 85] on which ENVY/Actra runs. Actor communication is an extension of standard Smalltalk message passing. This results in transparency between concurrent and sequential code and objects.

Actors overcome the limitations of the Smalltalk process model by providing an explicit mechanism for describing concurrent behaviour. The implementation of actors in Smalltalk is simple and straightforward and has all of the features required to realize an ABCL/R style model of concurrency. A basic actor is no more reflective than a typical Smalltalk object since its behaviour is described by its class. Under our scheme, classes continue to describe the structure of an actor, while meta-actors describe an actor's concurrent protocol.

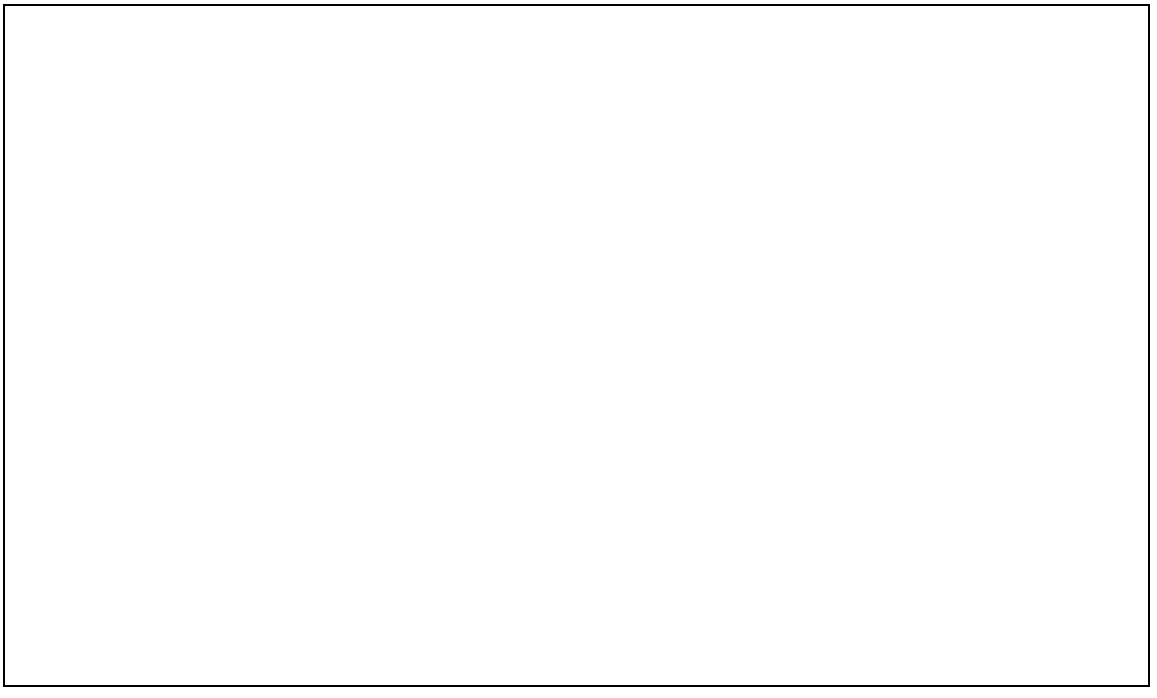

Figure 1: Actors and meta-actors

Figure 1 shows a concurrent meta-actor structure implemented in ENVY/Actra. Figure 1 shows two actors, their associated meta-actors and their communications patterns. Actors communicate by using standard Harmony messages to each other's meta-actors. The individual meta-actors dictate how messages are sent, received and processed. Three concurrent interactions are required to pass a message from B to A: One from B to ↑B, one from ↑B to ↑A and one between ↑A and A.

We have described a simple implementation of concurrent reflection facilities in terms of an industrial strength concurrent Smalltalk. The general use of reflection discussed here is not restricted in anyway to this implementation. Furthermore, it provides a basis for the use of reflection in real applications.

## Unified Distributed Simulation

The first application of our concurrent reflection capability was in the development of a new distributed simulation algorithm, Unified Distributed Simulation (UDS) [McAffer, 90a, 90b]. Typical distributed simulation algorithms are either completely synchronous or completely asynchronous. Unfortunately, real-world systems are not as well-behaved, and simulation models are forced to conform to the simulation tool. Using computational reflection and an object-oriented task model, we developed an algorithm to solve this problem.

The applicability of computational reflection in this domain was demonstrated in [Watanabe and Yonezawa 88], in which a simple ABCL/R implementation of the Time Warp [Jefferson 85] distributed simulation algorithm is described. The entire mechanism is implemented by redefining, in the meta-object, the way an object receives messages.

The basic idea of UDS is to let the actor receiving a message *decide* how the message should be handled. In UDS, actors use *variably asynchronous* message passing. That is, they have the capability to dynamically vary their own communication behaviour along a continuous spectrum from completely synchronous to completely asynchronous.

Each UDS actor defines two parameters, *risk* and *aggressiveness*. Risk is how far into the simulation's future the actor is willing to send a message while aggressiveness is how far into the simulation's future the actor is willing to accept a message. These two parameters control message reception and initiation for the actor and are controlled by its meta-actor. The meta-actor watches the flow of messages and the relevant system state, and sets the parameters accordingly. The reader is referred to [McAffer 90a] for a full discussion of this messaging scheme and a proof of temporal consistency.

Asynchronous interaction increases as risk and aggressiveness increase. Setting both to $\infty$ for all actors makes a completely asynchronous system. Conversely, setting all to zero makes a completely synchronous system.

To illustrate where UDS would be useful, consider a traffic system containing highways, city streets, intersections, crosswalks, and cars. When cars are on the highway, they tend to be asynchronous while cars on city streets must synchronize with intersections, crosswalks and slow moving cars. As a car moves from a city street to a highway it should adapt to the new concurrent environment. If it does not, it will restrict the *free flow* of the surrounding cars by forcing unnecessary synchronization. Reflective capabilities make this possible.

## A more general application

The ideas of concurrent reflection and risk and aggressiveness have a more general application, that of distributed task scheduling. Work done at Mitre [Bensley, *et al* 88] has resulted in a message stamping scheme which allows generic object-oriented programs to be run on a system using the Time Warp distributed simulation algorithm.

Thus, the Time Warp algorithm is used as a task scheduler. However, Time Warp only models completely asynchronous interaction.

The same message stamping scheme can be used for UDS. By applying the UDS algorithm in an analogous manner, we can effect variably asynchronous behaviour for arbitrary tasks. Scheduling responsibilities are distributed throughout the system eliminating the need for a central scheduler. Concurrent entities become reflective and thus flexible and adaptive.

This technique has yet to be implemented but is expected to provide many interesting insights to control and scheduling in a distributed environment. Such a process scheduler would also be useful for multiprocessor debugging and performance monitoring.

**Future avenues**

Reflection is only useful if it can modify behaviour rather than simply observe it. We have described a synchronization scheme which allows the operation of concurrent entities to be dynamically modified. Meta-actors can both observe and modify the behaviour of actors. There are several problems to be overcome before the UDS concept, and reflection in general, can be used in generic concurrent systems.

- What do we do with this capability?

- To what values should the aggressiveness and risk be set?

- When should they be changed?

- Can we build desired behaviours using this model?

As a first step to dealing with these questions, we propose that concurrent reflective computation be supported by small expert systems. Each meta-actor should have an expert system which monitors the actor and, according to some set of rules, changes the actor's operation. Behaviour is then described in a declarative way, which can itself be modified by the meta-meta-actor. This may not, in the end, be the right answer but it will allow us to set out, in a logical form, the rules by which we think actors should act.

Our experiences with concurrent reflection have been very positive. We found that in a sea of messages and interrupts, reflection allows tasks to reason about their environment and make better decisions regarding their behaviour. We believe that reflection has a significant place in real-world applications, especially those involving concurrency.

**References**

Bensley, E.H., *et al* (1989), "An Execution Model for Distributed Object-Oriented Computation", in *Proceedings of OOPSLA '88*, San Diego, CA, pp. 316-322, September.

Cointe, P. (1987), "Metaclasses are First Class: The ObjVlisp Model, in *Proceedings of OOPSLA '87*, Orlando, FL, pp. 156-167, October.

Gentleman, M. (1985), "Using the Harmony Operating System", ERB-966, NRCC No. 24685, National Research Council of Canada, Ottawa, Ontario.

Jefferson, D., (1985), "Virtual Time", *ACM Transactions on Programming Languages and Systems*, 7(3), July.

McAffer, J. (1990a), "Unified Distributed Simulation", M.C.S. Thesis, School of Computer Science, Carleton University, Ottawa, Ontario, March.

McAffer, J. (1990b), "A Unified Distributed Simulation System", in *Proceedings of the 1990 Winter Simulation Conference*, pp. 415-422, New Orleans, LA, December.

Watanabe, T. and Yonezawa, A. (1988), "Reflection in an Object-Oriented Concurrent Language", in *Proceedings of OOPSLA '88*, San Diego, CA, pp. 306-315, September.