An Actor-Based Metalevel Architecture for Group-Wide Reflection

Takuo Watanabe*
Department of Information Science
Tokyo Institute of Technology
JSPS Fellow (DC)
`takuo@is.s.u-tokyo.ac.jp`

Akinori Yonezawa
Department of Information Science
The University of Tokyo
`yonezawa@is.s.u-tokyo.ac.jp`

Abstract

The notion of *group-wide reflection* is presented. Group-wide reflection, a dimension of computational reflection in concurrent systems, allows each computational agent (actor/object/process) to reason about and act upon not only the agent itself, but also a group of agents which may contain the agent itself. Global properties of the group can be dynamically controlled through group-wide reflection. We have developed a simple yet general model for group-wide reflection based on the Actor model[?]. An operational semantics of a group of object-level actors is represented by another group of actors (a group of metalevel actors), which is an implementation of a transition system of the object-level group. We prove that the metalevel group correctly represents the operational semantics of the group in terms of transitions of configurations. Furthermore, migration of an actor from node to node is described as an example of group-wide reflection.

Keywords

reflection, group-wide reflection, actor model, object-oriented concurrent programming, object-group, ABCL/R, ABCL/1

1 Introduction

Our research goal is to construct a solid basis for programming adaptive/self-evolving computational systems. We believe that object-oriented concurrent computation with reflection provides a suitable framework for this goal.

The motivation of group-wide reflection is to cope with the following issues.

*Contact Address: Department of Information Science, University of Tokyo, 7-3-1, Hongo, Bunkyo-ku, Tokyo, JAPAN, 113, TEL/FAX: +81-3-5689-4365

1. *Object Group and Group Communication*:
   The notion of *object (process) group* offers a powerful abstraction mechanism for construction of large systems. Using a metalevel architecture, we like to make experiments on mechanisms such as *group communications*[?], dynamic (run-time) construction/destruction/fusion and migration of groups, and so on.

2. *Dynamic Modification of the Semantics of Message Delivery*:
   It is sometimes useful to have different semantics of message delivery in different groups of objects. If we want to implement the group-communication stated above efficiently, group-wide adaptation of message delivery semantics is often required. For example, consider the situation of using Timewarp mechanism[?] in a group. Changing the order of messages delivery by their timestamps makes it possible to have an efficient message scheduling which avoids frequent rollbacks and anti-messages.

3. *Maintaining Constraints among Objects*:
   Mutually constrained objects form a group. To maintain the constraints among them, controlling metalevel properties often offers a better means than ordinary message passing among the objects.

4. *Modeling Implementation-Related Features*:
   In an actual multicomputer concurrent system, there are some limited ways to obtain/modify global information of its subsystems. For example, in a coarse grained multi-processor system (*i.e.*, more than one object/process may run on each processor in a pseudo-parallel manner), we can obtain and/or control various global dynamic characteristics of objects/processes running in a single processor (*e.g.*, execution scheduling, allocation of resources, control of the messages traffics and so on). In other words, the machine-boundary of a processor inherently forms a group. Of course, such operations on the group have been utilized in actual systems (especially, operating systems), but they are usually introduced in either an ad-hoc or quite limited way ([?, ?]). We need a good formalism for uniform treatment of such implementation models.

In our previous approach to reflective computation in an object-oriented concurrent language ABCL/R[?], we adopted *individual-based* reflection: *i.e.*, each agent (object) can reason about and act upon the agent itself. In ABCL/R, a *meta-object*, which serves as the causally-connected metalevel representation of an object, is introduced for each object. A meta-object represents (implements) the structural/computational aspects of an object, and reflective computation is realized by sending messages to meta-objects. Since the computational activity in an object is sequential, technique for constructing a sequential reflective system can be used. This implies, however, each meta-object models only the local sequential aspects of an object. Thus, our previous approach is not sufficiently powerful to deal with the global information of a group of objects.

To realize a full-fledged computational reflection on a group of objects, a formal model of the group is needed. The model must represent the correct semantics of the group and should be accessible from members of the group. In addition, more importantly, the model and the group should be causally-connected: the model always represents (reifies) the current status of the group, and the changes made to the model should correctly reflect in the behavior of the group.

We use the *transition system* given by Gul Agha[?] for describing the operational semantics of a group of actors. In our approach, the transition system of a group is represented as another actor group which forms a concurrent meta-circular interpreter. We prove that it correctly represents the operational semantics of the group in terms of the transition of configurations.

2 Metalevel Architecture

In order to realize the group-wide reflection, we compose a group of actors and its metalevel description. The latter is also a group of actors. Suppose that S is a group of actors. We let $\uparrow S$ denote the group which forms a metalevel representation (meta-circular interpreter) of S. Actors in two groups can communicate with each other (*inter-level communication*). Since actors in $\uparrow S$ maintains the global information of S, we can access/modify the global metalevel information of a group through sending messages to actors in $\uparrow S$, and vice versa. Of course, the meta-circularity of $\uparrow S$ guarantees the causal connection between S and $\uparrow S$[?].

2.1 Configurations

Before presenting our metalevel description for an actor system, we briefly give a formalism for Actor model. It is basically the same as Agha's formalism. See [?] for more details.

An actor α is a pair $\langle m, \beta \rangle$ of its *mail address* m and its *behavior* β. The domain of actors $\mathcal{A}$ is defined as $\mathcal{A} = \mathcal{M} \times \mathcal{B}$, where $\mathcal{M}$ and $\mathcal{B}$ are the domains of mail addresses and behaviors, respectively. Note that in one system, no two actors have the same mail address.

A *task* is a triple $\langle t, m, k \rangle$ of a tag t, a mail address m and a message value k. A task represents a message which has been sent to an actor (a target actor) whose mail address is m, but which has not been received. The tag t is needed to distinguish two tasks having the same target address and the same message value The domain of tasks is defined as $\mathcal{T} = \mathcal{I} \times \mathcal{M} \times \mathcal{V}$, where $\mathcal{I}$ and $\mathcal{V}$ are the domains of tags and values used in messages. $\mathcal{V}$ is the disjoint sum of domains of mail addresses and other first class primitive values (such as numbers, lists, etc.).

The computation carried out by α in response to a message is described as the result of function application $\beta(t, k) = \langle T, A, \beta' \rangle$, where T is the set of tasks created by α, A is the set of actors created by α, and β' is the replacement behavior. The domain of behaviors $\mathcal{B}$ is defined as:

$$\mathcal{B} = \mathcal{I} \times \mathcal{V} \rightarrow F_s(\mathcal{T}) \times F_s(\mathcal{A}) \times \mathcal{B}$$

where $F_s(X)$ is the set of all finite subdomains (subsets) of X.

Let S be a system composed of actors. A *configuration* C represents a computational state of S at a certain frame of reference. This is represented by a pair $\langle \mathcal{A}(C), \mathcal{T}(C) \rangle$, where $\mathcal{A}(C) \in F_s(\mathcal{A})$ is a finite set of actors in S (called the *population* of C) and

$\mathcal{T}(C) \in F_s(\mathcal{T})$ is a finite set of *tasks* in S. Note that $\mathcal{A}$ and $\mathcal{T}$ are the domains of actors and tasks respectively. We let Γ_S denote the set of all configurations of S.

Computation in S is modeled as transitions between configurations in Γ_S. When a task $\langle t, m, k\rangle \in \mathcal{T}(C_1)$ is processed in $C_1 \in \Gamma_S$, the transition which results in a new configuration C_2 is denoted by:

$$C_1 \xrightarrow{\langle t,m,k\rangle} C_2$$

The configuration C_2 represents the situation where the actor having mail address m has just finished its computation for the task $\langle t, m, k\rangle$. If the target actor is in S, the following holds.

$$\begin{array}{l}\langle t, m, k\rangle \in \mathcal{T}(C_1) \wedge \exists\beta \in \mathcal{B}.\ s.t.\ \langle m, \beta\rangle \in \mathcal{A}(C_1)\\ \mathcal{A}(C_2) = (\mathcal{A}(C_1) - \{\langle m, \beta\rangle\}) \cup A' \cup \{\langle m, \beta'\rangle\}\\ \mathcal{T}(C_2) = (\mathcal{T}(C_1) - \{\langle t, m, k\rangle\}) \cup T'\\ where\ \ \beta(t, k) = \langle T', A', \beta'\rangle\end{array}$$

When the target is in another system S', then the configuration C_1' of S' will change to C_2'.

$$\begin{array}{l}\mathcal{A}(C_2) = \mathcal{A}(C_1), \mathcal{T}(C_2) = \mathcal{T}(C_1) - \{\langle t, m, k\rangle\}\\ \mathcal{A}(C_2') = \mathcal{A}(C_1'), \mathcal{T}(C_2') = \mathcal{T}(C_1') \cup \{\langle t, f(m), f(k)\rangle\}\end{array}$$

The function $f : \mathcal{M} \rightarrow \mathcal{M}$ translates a mail address in S to one in S'.

We write $C_1 \xrightarrow{*} C_k$ $(k \geq 1)$ when the following holds:

$$\begin{array}{l}C_1 \xrightarrow{\tau_1} C_2 \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_{k-1}} C_k\\ \text{where } \exists\tau_i \in \mathcal{T}(C_i)\ (i = 1, \cdots, k-1)\end{array}$$

The sequence above is called a *possible transition sequence.*

2.2 Metaconfiguration

Now we construct a causally connected metalevel representation for an actor system S. The metalevel representation for S, which is denoted by $\uparrow S$, is constructed as another actor system which implements the transition system of S. $\uparrow S$ is an actor system which represents the *concurrent* computational aspects of S, namely, $\uparrow S$ represents an operational semantics for S. In our model, $\uparrow S$ is also used as a meta-circular interpreter for S. This implies that the causal-connection between S and $\uparrow S$ is automatically guaranteed.

We show that $\uparrow S$ correctly models the semantics of S in terms of transitions between configurations. Notice that $\uparrow S$ to be constructed here is just one case of many possible metalevel representations.

As seen above, to define an actor system, it is not enough to pick up actors in the system — we need to define its configuration. We now define a special configuration of $\uparrow S$, called a *metaconfiguration.* A metaconfiguration is a configuration which *models* a particular configuration of S. We let $\uparrow C$ denote a metaconfiguration which models a configuration $C \in \Gamma_S$. We assume that the initial configuration of $\uparrow S$ is always a metaconfiguration. Below, $\Gamma_{\uparrow S}$ denotes the set of all possible configurations of $\uparrow S$.