Reflection for extending OO systems with distribution capabilities

Lourdes Tajes Martínez, Fernando Álvarez García, Marián Díaz Fondón, Darío Álvarez Gutiérrez, Juan Manuel Cueva Lovelle and Alberto Rodríguez García

Universidad de Oviedo
Department of Computer Science, University of Oviedo
Calvo Sotelo, s/n, 33007 OVIEDO-SPAIN
Phone: +34-8-5103368 Fax: +34-8-5103354
email: {tajes, falvarez@correo.uniovi.es}
email: {fondon, darioa, cueva@pinon.ccu.uniovi.es}

Abstract

The design of an adequate computational model for true distributed objects is a non-trivial task, mainly if we want a dear distinction between mechanisms and policies and even more if we want policies to be easily replaced or customised. We propose to build an integral object-oriented system structured as an object-oriented operating system based in an object-oriented abstract machine and reflection as the glue that allows interaction between them.

Keywords: Integral Object-Oriented System, Object-Oriented Operating System, Object-Oriented Abstract Machine, Distribution, Flexibility, Reflection.

1. Introduction

With the Internet boom and distributed systems, an heterogeneous distributed interoperable object environment is appearing. Platforms and architectures such as Java[6] and CORBA[9] are responsible for the evolution towards this environment, increasingly seen as an environment for the future.
An integral object-oriented system (IOOS) is an example of this *world of objects* environments. It provides a virtually infinite space where objects live indefinitely and exchange messages regardless of their location. Oviedo3[3] is an IOOS structured around a minimal kernel, implemented as an object-oriented abstract machine (OOAM), extended with an object-oriented operating system (OOOS), a set of normal, replaceable objects that will extend it with distribution characteristics. Our aim is to explore reflection for allowing OOAM-OOOS collaboration in distribution issues.
The rest of the paper is organized as follow. Sections 2 and 3 present the OOAM and the OOOS. In section 4, the computational model is described. Reflection is presented in section 5 and sections 6, 7 and 8 present the way these concepts are applied to the IOOS. In section 9 related work are presented and next, the advantages of adopting a reflective architecture to get distribution capabilities in a flexible way are shown. Finally, some implementation issues and the conclusions are presented.

2. The object-oriented abstract machine

The **OOAM** provides low level support for objects and a default implementation for the most basic functionality allowing object creation, deletion, communication (method invocation) and exceptions. The overall organization of the OOAM is shown in figure 1.

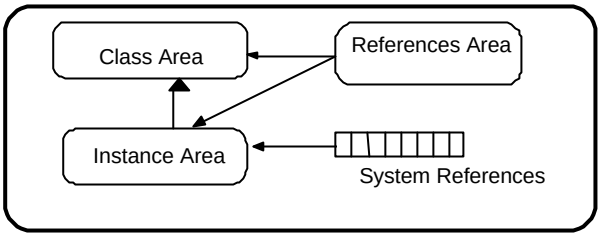

Figure 1: Reference architecture for the object-oriented abstract machine.

Other abstract machines could also be adapted for this purpose. Indeed, due to the popularity of the Java language, and the availability of public specifications, the adaptation of the Java Virtual Machine [6] is under consideration.

3. The object-oriented operating system

The **OOOS** modifies and extends the minimal functionality provided by the OOAM in areas like persistence, distribution and concurrency.
In a first stage, the OS is designed as an **OO framework**. That is, a class hierarchy where each class offers some specific OS functionality is built. A lot of advantages derive from this approximation as code reuse or the improvement of portability.
Besides, the functionality of **the OS will be implemented as a set of normal objects at runtime**. So, the OS will extend the OOAM in a transparent way.

4. Basic computational model supported by the abstract machine

The OOAM adopts an **active object model**. It defines the object abstraction as an autonomous execution entity with a set of virtual processors. The OOAM default implementation for objects is as sequential entities, so inner concurrency is controlled.

4.1. Runtime objects

At runtime, inner representation of objects includes state, methods and the execution area, a set of **threads** (inner machine objects) that store information about methods being executed: stopped, started, delayed or postponed methods (see figure 2).

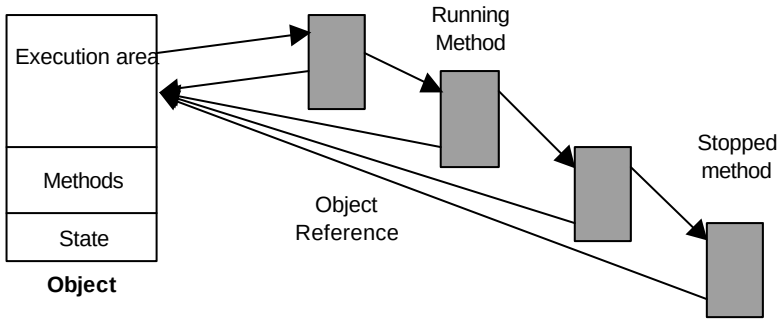

Figure 2: Object execution environment

4.2. Communication between objects

The OOAM offers a set of instructions for method invocation implemented as message passing. The synchronous and asynchronous message passing instructions inform objects that the execution of one of their methods is requested.

5. Reflection

Reflection is the means for the OOAM – OOOS interaction, so we are presenting it before to describe how it is applied to the IOOS. The most extended definition for reflection is [BGW93]: *"Reflection is the ability shown by a program for manipulating, as data, the representation of its own state during execution"*.

5.1. Base system and meta-system

A computational system can be viewed as composed of two systems. A **base system** solves the external problem describing the computation from the application point of view. A **meta-system** maintains information about base-entities and describes the way the computation in the previous level is carried out by means of meta-entities [7].

5.2. Meta-system model or meta-model

The key idea in reflection is to access and modify the meta-system, changing the way it performs the actions. But, as direct manipulation is not allowed, a restricted representation of the meta-system, named **meta-model**, is built (**Reification**)[7]. The meta-model can be accessed (**Introspection**) and manipulated (**Intercession**) and any change in the meta-model is reflected in the meta-system behaviour (**Reflection**).
For introducing reflection in an OO system in an uniform way, we propose to describe the meta-model using the OO paradigm.

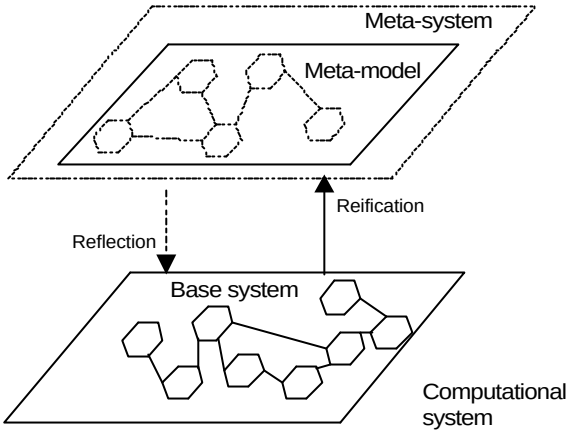

Figure 3: Meta-system description by means of an object meta-model.

5.3. Structural and computational reflection

Reflection can be expressed in two ways: *structural* and *behavioural* or *computational reflection.* The *Structural reflection* reifies structural aspects of a program like inheritance. An example is the Java Reflection API [8]. *Computational reflection* defines the environment where the base actions are executed.

6. Reflection to get a flexible computational model for distribution

The OS objects will collaborate with the OOAM extending it with features as distribution. A meta-model of the machine is built where meta-objects expose, as normal objects, inner machine objects. A set of meta-objects is attached with application objects adapting the meta-model to their own requirements. These meta-objects can be implemented by machine objects or by OS objects if the application replaces the default meta-object modifying the default behaviour of the OOAM.

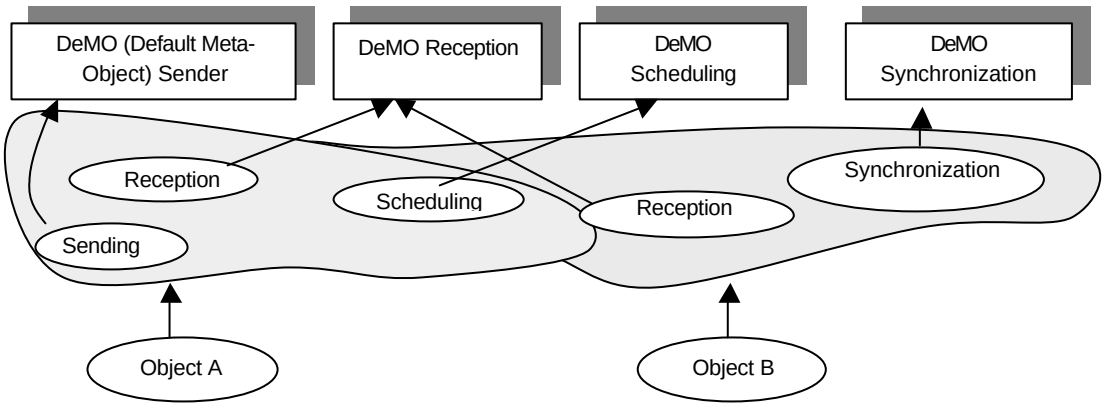

Figure 4: Meta-space attached to base objects: OOAM objects OOOS objects.

6.1. Structural Reflection

Structural reflection exposes aspects of the OOAM architecture and the structure of the objects at runtime. Instances from these classes allow introspection and intercession.

– **Expose the inner architecture**. Reifying the OOAM areas by means of instances of ClassArea, InstanceArea, ReferenceArea and ThreadArea, allows finding out the elements stored in the reflected areas as well as storing new elements at runtime.

- The **composition of objects at runtime** is reified by means of Class, Instance and Method. They allow managing information about the internals of classes (methods or aggregates of a class), instances (current values of the attributes) and methods (its arguments or the exceptions it raises) as well as creating new ones at runtime.
- The **runtime environment** is exposed by means of Thread objects that allow knowing and acting upon the current threads modifying their priority or state or accessing the local variables or the call/reply chain that conduces to this thread. Other systems like Smalltalk [10] shares this idea.
- The **meta-space** is exposed by means of MetaObject allowing an object to ask its meta-objects about the way they implement its functionality as well as replace one with another that implements the same functionality in a different way.

7. Behavioural reflection

Finally, we propose to reify the following aspects of the behaviour of the OOAM for allowing the base level accessing and modifying them[4].

7.1. Object communication

Sending and receiving messages and the message acceptation policy is exposed by two independent meta-objects: sender and receiver. The sender controls the actions to carry out before and after sending the message. The receiver decides about the message reception such as delaying method execution, delegating or rejecting the message.

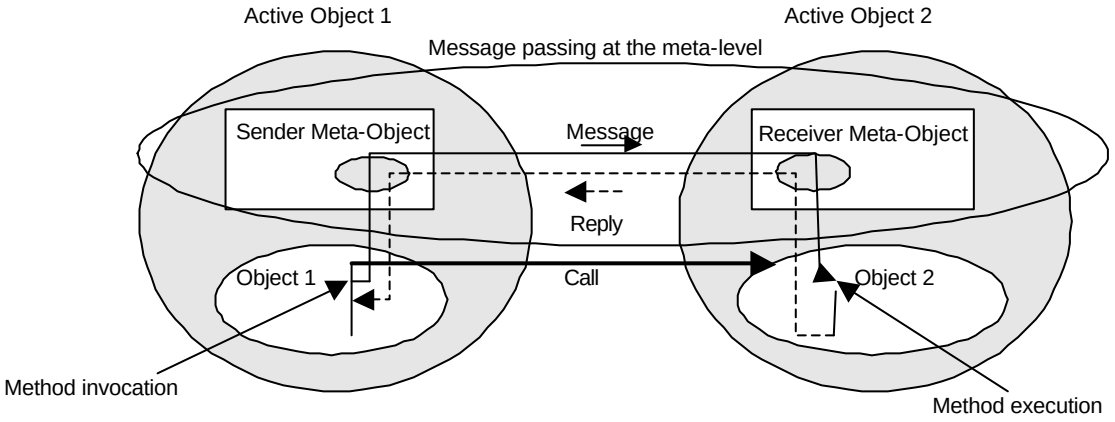

Figure 5: Composición Interna de un Objeto.

7.2. Inner concurrency control

As objects are defined as concurrent autonomous agents some synchronization policy is needed to determine the adequate conditions to activate the execution of a method. The basic idea shared with other jobs as [2] is to divide the state of the concurrent object in a sequential side and a concurrent side as shown in figure 6.

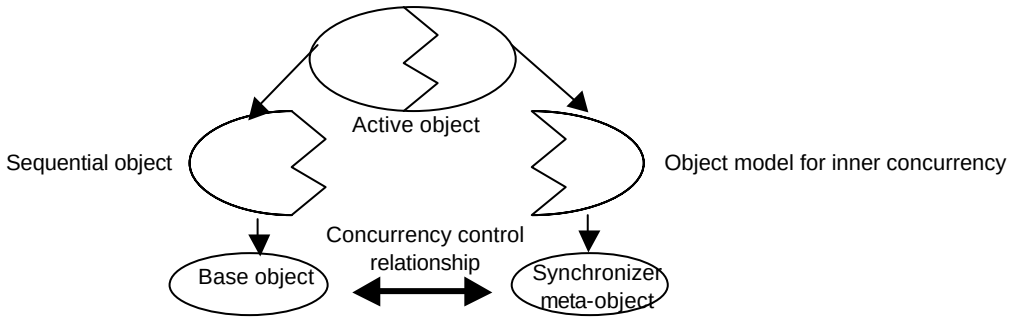

Figure 6: Controlling inner concurrency at the meta-space.

The sequential side implements the functional code. The concurrent side implements object behaviour to synchronize concurrent execution of its methods. This concurrent side, implemented as a normal object, is named Synchronizer.

7.3. Scheduling

In a concurrent system a scheduling policy is needed to determine the execution order of the tasks. The meta-object scheduler implements the scheduling policy and can be replaced with other that implements another different policy.

7.4. Base-level / meta-level interaction

The main event that provokes transferring control to the OS is message passing. The OOAM can execute the message passing by itself or transfer control to a meta-object.
Other events that provokes transference control to the meta-level are those related with method execution, that is, start, stop or finish a method. These events change object state and synchronizer meta-object must actualize it.
Lastly, time interrupts are implemented for scheduling. When the machine produces a time interrupt, the scheduler meta-object executes its methods.

8. Distribution: An example of OOAM – OOOS reflective collaboration

Although distribution transparency is important for simplifying most of the applications, some of them need some control over distribution, so facilities for distribution control must be provided. Some meta-objects, specially sender and receiver, allow customizing some aspects of the remote method invocation, migration and other.

8.1. Remote method invocation

Sender and receiver meta-objects will be modified to extend the local method invocation provided by the OOAM to a remote one. The OOAM gives control to the sender meta-object each time an invocation takes place.
If both the invoked object and the invoked method exist locally (introspection over InstanceArea and Instance objects) the receiver meta-object of the invoked object is informed to carry out the server part of the invocation.
If the invoked object does not exist locally, it must be located. The locator OS object is invoked to obtain the location information needed to forward the request. A message containing the invocation information is sent to the destination machine by means of a communicator object that deals with the network particularities. A peer communicator object in the destination machine receives the message and creates a worker object, which converts a remote method invocation into a local one.

8.2. Object migration

When talking about object migration, some questions must be answered. Policy issues like when, how much and where to move, and the mechanism itself, how to move.

8.2.1. Policy issues

The policies about object migration are implemented by means of a set of OS objects that collaborate with the OOAM deciding how to implement some aspects of the migration, basically where and when to move and what objects to move.

8.2.2. The migration mechanism

It consists on the set of operations, transparent to base-level objects, that results in the disappearance of an object in the origin machine and its appearance in the destination. The mover OS object will be invoked any time an object must be moved and it will execute the following operations to complete the movement:

1. It obtains, in collaboration with the synchronizer, a copy of the object where no threads are in the running state and arriving invocation requests are blocked. Then, the mover asks the InstanceArea for serializing the object's state and computation.
2. The local communicator object sent the string copy to the destination machine. A remote communicator will receive the message and gives it to a worker object.
3. The worker object asks the InstanceArea object to create an object from the string.
4. Finally, the original copy is deleted, the object location information is updated and it is reactivated, that is, the synchronizer is invoked to unblock every pending invocation requests and to resume every thread that was in a running state.

As it could be seen, object migration is supported by both the structural and behavioural reflection. The OOAM and the object been moved are unaware of the migration operation, that takes place at the meta (OS) level.

9. Related work

There are a number of OOOS, such as Choices, Apertos, Spring and others. Each one is focused on a different OO philosophy with different structures. Choices [1] is based in an OO C++ framework which can be specialised to give customised OS. Apertos [11] uses an object/metaobject separation to structure the OS. It structures its OS around a set of meta-spaces which causes a performance penalty. Spring [5] uses an interface definition language to define the system interfaces as objects and has a microkernel with the distribution abstractions embedded.

10. Advantages of choosing reflection as the base of an IOOS based in an OOAM and an OOOS

Flexibility is the main advantage obtained offering the OS mechanisms as extensions of an OOAM. Each object can instance its own meta-objects. They modify the standard behaviour of the OOAM adapting it to specific requirements.

Uniformity is maintained because the OS mechanisms are offered in an OO way.

The active object model is a powerful help to the distribution mechanisms because it provides a compact view of the object what makes computation **transparent** to object distribution and migration mechanism. No special operations to distribute computation need to be developed.

11. Current implementation status

Several prototypes for endowing OOAM with reflection have been finished. The first one has implemented dynamic structural reflection, so classes and instances can be modified at runtime. The second one is

finishing to implement an OOAM endowed with behavioural reflection. It mainly reifies object communication and inner concurrency.
They both implement reflection by means of a class hierarchy, written in the assembly language, that reifies the OOAM architecture and inner structure of instances as shown in figure 7.
In the distribution area, a prototype has also been finished, but with the distribution policies and mechanisms at the machine level. We are now working on a full reflective machine to meta-implement the distribution issues presented in this paper.

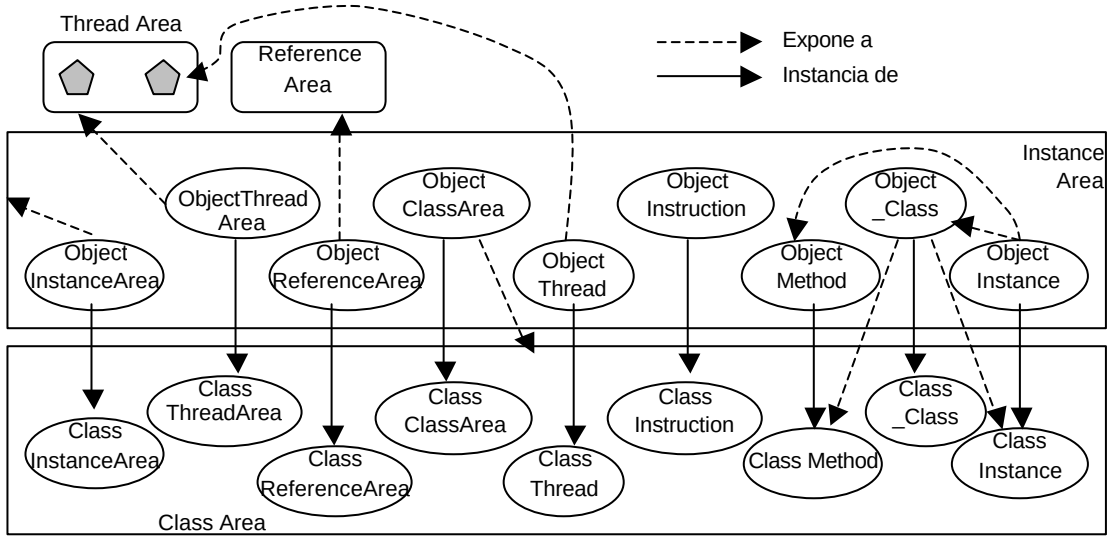

Figure 7: A set of basic classes is the kernel for structural reflection.

12. Conclusions

An Integral Object-Oriented System (IOOS) is a system based solely on the object-oriented paradigm, which provides a world of objects environment. Oviedo3 is an IOOS based on an object-oriented abstract machine extended with an object-oriented operating system.
Distribution is an important property that must be present in these environments. Where to introduce distribution support in the IOOS is not trivial if we want to be able to customize distribution to specific needs. An immediate solution could be the ad-hoc extension of the OOAM with distribution properties. This is an inflexible solution.
A better solution is to endow the abstract machine with a reflective architecture that divides the world of objects in base (application) objects and meta (OS) objects. The OO paradigm is not broken and object modifying can easily adapt OS services.
Remote method invocation and object migration are solved at the meta-level by means of a set of meta-objects that write an optional mechanism for them.
While preserving transparency, this reflective architecture is flexible enough as to also permit special applications to control distribution issues by means of a set of meta-objects exposed for this purpose.

References

[1] R.H. Campbell, N. Islam and P.W. Madany. *Choices, Frameworks and Refinement*. Computing Systems , 5(3):217-257, 1992

[2] D.Caromel and J. Vayssière. *A Java Framework for Seamless Sequential, Multi-Threaded, and Distributed Programming*. Proceedings of the ACM Workshop Java for High-Performance Network Computing. Pp. 141-150. 1998.

[3] J.M.Cueva Lovelle, R.Izquierdo Castanedo, D.Álvarez Gutiérrez, L. Tajes Martínez, M.A.Díaz Fondón, F.Álvarez García and A.B. Martínez Prieto. *Oviedo3: Acercando las tecnologías orientadas a objetos al hardware*. (In spanish) I Jornadas de trabajo en Ingeniería de Software. Sevilla. 1996.

[4] B. Gowing and V. Cahill. *Reflection + Micro-Kernels: An Approach to Supporting Dynamically Adaptable System Services*. CaberNet Radicals '96. Connemara, Ireland. 1996.

[5] G.Hamilton and P.Kougiouris. *The Spring Nucleus: A microkernel for objects*. In Proceedings of the 1993 Summer USENIX Conference. 1993.

[6] D.Kramer, B.Joy and D.Spenhoff. *The Java™ Platform White Paper*. JavaSoft, May 1996. URL: ftp://ftp.javasoft.com/docs/JavaPlatform.ps

[7] P.Maes. *Issues in Computational Reflection*. Meta-Level Architectures and Reflection. Pp. 21-35. P.Maes and D.Nardi, Eds. Elsevier Science Publishers B.V. 1988

[8] C.McManis. *Take and in-depth look at the Java Reflection API*. JavaWorld, September, 1997.

[9] Object Management Group. *The Common Object Request Broker: Architecture and Specification, revision 2.3*. June, 1999. Available by HTTP in URL: http://www.omg.org.

[10] F.Rivard. *Smalltalk: A Reflective Langage*. In Proceedings of the Reflection'96. G.Kiczales, ed.

[11] Y.Yokote. *The Apertos Reflective Operating System: The Concept and Its Implementation*. Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'92). Pp. 414-434. A.Paepcke, ed. Also SIGPLAN Notices 27(10), October 1992