Object Distribution Subsystem for an Integral Object-oriented System

Fernando Álvarez García, Lourdes Tajes Martínez,
María Ángeles Díaz Fondón, Darío Álvarez Gutiérrez, Juan Manuel Cueva Lovelle
Department of Computer Science, University of Oviedo
Calvo Sotelo, s/n, 33007 OVIEDO - SPAIN
{falvarez, lourdes, fondon, darioa, cueva}@pinon.ccu.uniovi.es
Phone: +34-98-5103370 Fax: +34-98-5103354

Keywords: object distribution, object-oriented abstract machine, object-oriented operating system

Abstract

Object distribution is an essential feature for any object-oriented operating system. In this paper we show how to build the distribution subsystem for an integral object-oriented system based in an object-oriented abstract machine. We propose a distribution mechanism in which the abstract machine intervention has been taken to a minimum. An abstract machine will only know and manage its local objects. Other distributed concepts like replication and transactions can be built above this basic distribution layer.

This work has been partially supported by the second regional plan (FICYT) of research of the Principality of Asturias, Spain, under project PBP-TIC97-01 "Object Oriented Integral System: Oviedo3"

1. Introduction

Oviedo3 [Cueva, 1996] is a research project that tries to build an experimental integral object-oriented system where every component, such as user interfaces, applications, languages, compilers, databases, etc. and the operating system itself share the same object-oriented paradigm.

In this paper we focus in the distribution subsystem of SO4, the Oviedo3 object-oriented operating system. SO4 is build above the Carbayonia object-oriented abstract machine. Carbayonia is a high level abstract machine which gives the object basic support for the whole integral system. Both the abstract machine and the operating system give their services by means of a set of objects. Homogeneity among abstract machine, operating system and user objects is one of the main goals of the project.

We believe SO4 should supply transparently the following features for the objects of this integral system: a capability-based protection mechanism, complete persistence, object distribution and concurrency. In AGRA, the distribution subsystem of SO4, we define the object as the distribution unit. An object will be an active entity that encapsulates its state, behavior and computation (multi-threaded). The responsibilities of the distribution subsystem go beyond object mobility and remote method invocation. It must support features like encapsulation, inheritance and polymorphism along the distributed system.

Next section introduces the OO abstract machine in which the operating system is based. Then some preliminary advantages due to the machine are listed. The master guidelines of the OS designed to

support the integral system are described in section 3. Section 4 considers a number of possible mechanisms of interaction between the operating system and the machine, specially reflectivity. Section 5 describes de distribution subsystem in terms of its features and the set of operations that must be redesigned at the operating system level to achieve the maximum degree of transparency. Finally, section 6 describes how this architecture embodying the reflective OO abstract machine and OS support objects yields a very flexible integral OO computing environment.

2. The Object-Oriented Abstract Machine

Using an OO abstract machine as the substrate for structuring the OOOS has many advantages. This machine supporting the OS should supply the essence of the aspects mentioned above: a simple self-contained homogeneous object model, with the object identity used as the unique access reference. This will be the only system abstraction.

Next, the overall organization of the Carbayonia[1] abstract machine is shown. It has been designed specifically to support the OS as described in next section.

The machine supports an object model with the following features:

- Object identity and abstraction.
- Encapsulation.
- Inheritance, and also generic and aggregation relationships between objects.
- Polymorphism or message passing.

Fig. 1 shows the architecture of the Carbayonia abstract machine [Cueva, 1996], consisting of four areas: class area, references area, instance area, and system references area. Each area can be considered as an object in charge of the management of its data.

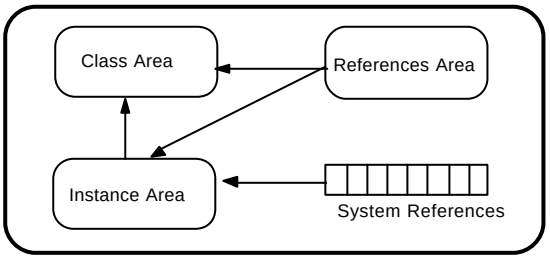

Figure 1. Architecture of the Carbayonia abstract machine.

- Class area. Maintains the description of each class. There is a set of primitive classes defined permanently in this area.

[1]"Carbayón" means "big oak" in Asturian, the local language. It is a symbol for the city of Oviedo. "Carbayonia" means land of oaks.

- References area. Stores the references. Every reference has a type (relates to the class area) and points to a object of this type (relates to the instance area).

- Instance area. Stores objects created. At run time, the information of its class can be accessed in the class area.

- System references area. Contains references with specific functions in the machine.

The main characteristic of the machine is that every action upon an object is made using a reference to it. The machine language is a pure OO low level language. It allows class declaration, method definition and exception handling. It will be the intermediate language used by compilers of programming languages like Eiffel, C++, etc. There are a number of classes built into the class area of the machine, with "Object" as the root class.

In the following, we will refer to objects in general, making no distinction between objects and classes.

Some of the preliminary advantages derived from using an abstract machine in general for this OS support task are:

- Portability. All system elements, from the OS to applications developed in it will work without modifications in every platform where the abstract machine is present.

- Impedance mismatch is removed. There is a common object model for the system, supplied by the machine for every object, including OS ones.

- Programming language independence. Once an object is created in the machine, it is independent from the programming language used to define it.

- Implementation of OS features. We will see how the distribution subsystem benefit from the use of the abstract machine.

3. The Object-Oriented Operating System

The heart of the Oviedo3 system is the operating system, named SO4[2] [Álvarez, 1997]. The OS offers the abstraction of a single object space where objects exist indefinitely, virtually infinite, and where objects placed in different machines cooperate transparently using messages. Besides, the OS transparently achieves a set of important features: security, persistence, distribution and concurrency.

SO4 can be classified as an object-support operating system but with also an object-oriented internal design. It is worth to note that both object models are the same.

The master guidelines of the design of SO4 are:

- Object as the only abstraction. The unique existing entity is the object, of any granularity.

[2] **S**istema **O**perativo para **O**viedo**3** (Operating System for Oviedo3).

- Intentionally “standard” object model, with the more common features found in OO programming languages: encapsulation, inheritance, and polymorphism [Booch, 1993].

- Exclusively OO working mode. An object can only create classes inheriting from others, create objects from a class, and send messages to other objects.

- Simpleness. To adhere to the above guidelines, whilst achieving maximum simpleness.

These are some important characteristics we wish to impose on objects, in accordance with these guidelines:

- Homogeneous objects. All objects, including OS ones, are treated the same. There are no special objects. OS objects themselves are not different from other objects.

- Transparency. Objects must not be aware of the existence of OS mechanisms to achieve features as persistence, message passing, distribution, etc.

- Self-contained objects. All the information about an object is encapsulated in its state, including computation, thus favouring an active object model. The behaviour must not be dependent on other objects. This is conceptually simpler, and makes easier to obtain transparent persistence, distribution, etc. mechanisms.

- Complete semantics. Objects have all the semantics embedded in the object model. They are not considered just as a contiguous memory space.

- Object identity. Each object has a unique identifier, which used as a reference is the only means to access it. Objects have no physical address, nor exists the concept of memory address space.

4. Abstract Machine and OS Relationships. Reflectivity

The abstract machine can be considered as a kernel for the OS, supplying a basic support for objects, which will be extended by the OS to incorporate additional features. The machine should supply mechanisms to accomplish this. We consider three mechanisms basic for this, which should be present in the machine. These mechanisms can be used alone or in combination:

- Internal modification of the machine. To adapt the inner workings of the machine to ease the implementation of a feature.

- Extension of the functionality of the basic classes. To add new attributes and methods to enlarge the functionality of the basic classes.

- Reflective architecture. Collaboration in the working of the machine. To Allow OS objects to collaborate with the machine when it cannot accomplish a task by itself by means of a reflective architecture [Maes, 1987]. The machine should be given an architecture where the machine objects are exposed as normal objects. Thus, machine objects can be used by normal objects, and can themselves use this normal objects. When a machine object can not continue for some

problem, it raises an exception that activates an OS object. This object will collaborate with the machine, intercommunicating with its objects to solve the problem, treating them just as other normal objects.

5. AGRA: the Distribution Subsystem

On a distributed architecture consisting of several Carbayonia abstract machines, the distribution subsystem [Chin, 1991] allows inter-object communication without regard to its location, and provides mobility mechanisms with the maximum degree of transparency. Every object in the integral system can be considered for distribution and can accept remote invocations without doing any special action. The desired result is a distributed abstract machine consisting of a distributed instances are, a distributed class area, a distributed references area and a set of system references. The goals of the distribution subsystem [Cueva, 1996] can be reduced to:

- Access and location transparency [Coulouris, 1994]. To provide a location service that delivers the messages to the objects, regardless of their location.

- Object mobility and load balancing. Objects will move from one abstract machine to another in order to maximize the overall performance.

Some of the features of the abstract machine imply several immediate advantages for distribution:

- Unique object identifier. Access and location transparency implementation is easier because of this uniqueness. Objects are always referenced and accessed in the same way regardless of their localization.

- Self-contained objects. Moving an object between machines is achieved just by moving its encapsulated state that includes the computation. Conceptually, this fact notably simplifies the migration operation.

The abstract machine gives the minimum support for distribution: all it has to do is to raise exceptions that will be managed by distribution subsystem objects. Even object migration, object finding and high-level naming policies are implemented in separated objects that permit their modification.

When we move from a standalone abstract machine to a distributed environment, several basic operations will be affected and some others appear:

- Object creation. It is possible that the object that invokes the creation operation and the class definition of the object being created do not reside in the same abstract machine. We have to decide where to create the new instance. It is easier to create the object in the machine where the class resides, and after move the object to the best possible location (migration policy). This can also happen with the aggregates of the object, and everything works the same.

- Object removal. Deleting objects in a distributed environment is a very delicate operation. A set of objects at the operating system level will be responsible of this distributed garbage collection.

- Method invocation. The (local) abstract machine that has to execute the method invocation tries to resolve it locally (the object being invoked is in the local instance area or in the local persistence storage). If it is there, the invocation does not pose any problem. If not, the abstract machine raises a machine exception that will be captured by an operating system object. This object is the responsible of locating the target abstract machine (with the help of a location server object) and forwarding it the invocation. If such an object is not known for the abstract machine, the caller is returned with a No instance exception: we cannot make use of the distribution feature.

- Argument passing. Every argument of an invocation (locally or remotely resolved) is, like in CORBA [OMG, 1997], an object reference.

- User exceptions propagation. User level exceptions propagate from the called object to the caller with no regard of their respective locations. The messages sent back from the invoked object to the caller can hold return results of the method invocation or an exception not handled.

- Object mobility. When moving an object, we have to pack its internal state and its computation state in the source machine and unpack it in the destination machine (selected using the migration policy in use). We use the same pack and unpack mechanisms as the ones used by the persistence subsystem [Cueva, 1996]. Besides, the object being moved is labeled as moving in the instance area of the source abstract machine, so any invocation to the object must wait until it recovers its normal state in the destination machine. When the object has been placed definitively in the destination machine, the local moving object is removed.

The set of operations and features just described constitute the basic layer of the distribution subsystem. Other abstractions like replicated objects or fragmented objects [Shapiro, 1989] could be built above this basic layer for performance reasons. We are also thinking of endowing the distribution subsystem with a transaction-like mechanism [Taylor, 1994], with the aim of our system execute normally in an unreliable network.

6. Current implementation

The first version of the distribution subsystem has just been finished. All the operations mentioned before excepting object mobility have been implemented. The main problem for implementing object mobility is the execution model of the abstract machine. At present, it is a passive object model, that is, an object has not execution power and a different entity of the system (the thread) is the responsible of passing from object to object executing their methods. To support this execution model, a stack of contexts, each containing information about the current executing method, is used. The difficulty arrives when we want to move an object involved in one or more context stacks because it has some methods in an execution state.

In the active object model, the object owns all the execution power, and when it moves, its whole internal state, including computation, is moved. We are now working on endowing the abstract machine an active object model to facilitate object mobility.

The main goal of the first prototype of the distribution subsystem was not to modify the abstract machine interface. New instructions have not been defined nor the basic semantic of the existing ones

has been modified. Every Carbayon program executing correctly in the centralized machine, executes correctly in the distributed version, regardless the location of every object.

To make possible for the abstract machine to execute its local threads at the same time it receives remote requests and it is waiting for the results of a remote invocation, we have moved to a multi-threaded (underlying operating system threads; from here, OS-threads) abstract machine. One OS-thread is the responsible of executing local method invocations. When executing a local method, there could be the necessity of invoking remote methods. At this moment, a new OS-thread is created for preparing the remote call, sending the message to the destination machine, and waiting for the result. The first OS-thread would continue executing pending invocations. When the result arrives, the created OS-thread is waked-up and it permits the local stopped method to continue. Finally, an OS-thread in every machine is the responsible of handling the arriving invocations. Every thing it has to do is to put the arriving invocation in the pending invocations pool. Given that several OS-threads are executing in parallel, a semaphore-like synchronization mechanism is used to avoid inconsistencies in the abstract machine structures.

The Remote Procedure Call (RPC) mechanism is used to communicate different abstract machines in order to allow remote method invocation. The parameter list of a RPC call is very simple:

- The first parameter is the identifier of the called object.
- The second parameter is a reference to the invoked method.
- The rest of the parameters forms a list of references that will act as parameters for the invoked method.

We have not included any protection mechanism for the message in this first version.

This first prototype has been finished recently, so information about the overhead introduced by the distribution subsystem has not been evaluated yet.

7. Reflective abstract machine + OOOS: very flexible integral OO computing environment

The abstract machine provides an object model and the support for objects. To accomplish this, the machine is organized in a reflective way into a set of objects. OS objects give transparently additional features to the objects, like persistence, distribution, etc.

The reflective architecture of the machine makes machine objects usable just like any other objects. This yields a computing environment seen as a set of homogeneous interacting objects, regardless of being machine, objects implementing OS functionality or normal user objects. Thus the separation among base machine, operating system and user applications is removed: a single integral OO computing environment (fig. 2).

In this computing environment, the OS, user applications, and even the base machine itself, take advantage of the object-oriented paradigm, including areas such as flexibility, extensibility and reusability.

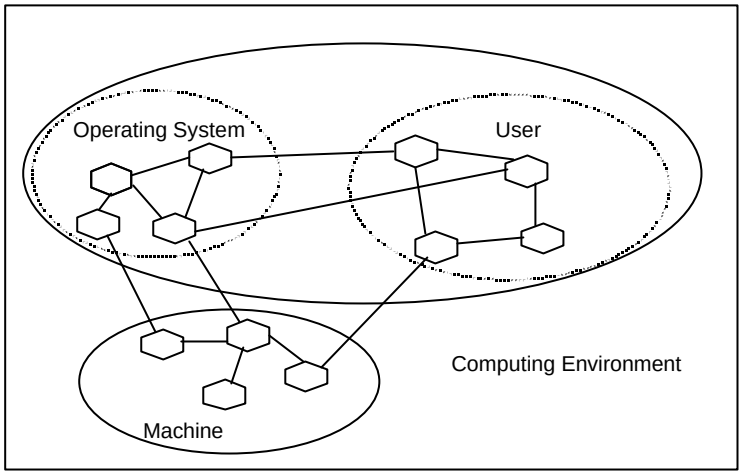

Figure 2. Computing environment composed of a set of homogeneous objects

8. Conclusions

The combination of an OO abstract machine offering object support with an OO operating system implemented as a set of objects is a promising way of structuring OO integral systems. Advantages such as portability, language independence and impedance mismatch removal are complemented with improvement and ease of implementation of the functionality of the OO OS.

A reflective architecture for the abstract machine is one of the most important mechanisms the machine must provide to qualify as a substrate for an OO operating system.

The most relevant aspects of the OOOS are security, persistence, distribution and concurrency. This paper is focused in the distribution subsystem of the SO4 operating system, called AGRA.

The distribution subsystem is responsible of the remote method invocation mechanism and the object migration facility. It has been designed for achieving the maximum degree of transparency for the rest of the system and to maintain one of the main design principles of the integral system: object homogeneity from abstract machine objects to user objects. Besides, we propose a distribution model where the participation of the abstract machine has been reduced to the raising of an exception.

We have shown that other distribution features like replication and fragmentation could be implemented above our design for adapting the environment to specific needs.

Finally, the first prototype of the distribution subsystem has been presented. It only lacks object mobility and this will be solved when the abstract machine change its execution model to an active one.

References

[Cueva, 1996] J.M. Cueva Lovelle, and others, Sesión Sistemas Operativos Orientados a Objetos: Seguridad, Persistencia, Concurrencia y Distribución (Object-Oriented Operating Systems: Security, Persistence, Concurrency and Distribution), II Jornadas sobre Tecnologías Orientadas a Objetos, Oviedo, Spain, March 1996 (in spanish)

[Álvarez, 1997] D. Álvarez Gutiérrez, L. Tajes Martínez, F. Álvarez García, M.A. Díaz Fondón, R. Izquierdo Castanedo, J.M. Cueva Lovelle, An object-oriented abstract machine as the substrate for an object oriented-operating system, Workshop on Object-Orientation and Operating Systems, 11th European Conference on Object-Oriented Programming (ECOOP'97), Jyväskylä (Finland), June 1997

[Booch, 1993] G. Booch, Object-Oriented Analysis and Design with Applications, 2nd edition, Benjamin Cummings, 1993

[Maes, 1987] P. Maes, Concepts and Experiments in Computational Reflection, Proc. OOPSLA, pp 147-155, 1987

[Chin, 1991] R.S. Chin and S.T. Chanson, Distributed Object-Based Programming Systems, ACM Computing Surveys, Vol. 23, N.1, March 1991

[Coulouris, 1994] G. Coulouris, J. Dollimore and T. Kindberg, Distributed Systems. Concepts and Design, 2nd edition, Addison-Wesley, 1994

[OMG, 1997] Object Management Group, The CORBA/IIOP 2.1 Specification, http://www.omg.org, 1997

[Shapiro, 1989] M. Shapiro, Y. Gourhant, S. Habert, L. Mosseri, M. Ruffin, C. Valot, SOS: an Object-oriented Operating System – Assessment and Perspectives, Computing Systems, 2(4) pp. 287-338, Dec. 1989

[Taylor, 1994] P. Taylor, V. Cahill and M. Mock, Combining Object-Oriented Systems and Open Transaction Processing, The Computer Journal, Vol. 37, No. 6, Aug. 1994, pp. 487-498