An Object-Oriented Abstract Machine as the Substrate for an Object-Oriented Operating System

Darío Álvarez Gutiérrez, Lourdes Tajes Martínez, Fernando Álvarez García,
María Ángeles Díaz Fondón, Raúl Izquierdo Castanedo, Juan Manuel Cueva Lovelle
Department of Computer Science, University of Oviedo
Calvo Sotelo, s/n, 33007 OVIEDO - SPAIN
{darioa,tajes,fag,fondon,ric,cueva}@pinon.ccu.uniovi.es

Abstract *Using an object-oriented abstract machine brings a number of benefits for the construction of an object-oriented operating system. In this paper we describe the structure of an abstract machine designed for this task. This machine provides the basic object model and support for the rest of the system. Among other options, we propose a reflective architecture as a collaboration mechanism between the machine and the OS. Finally, we show how using this architecture based on the abstract machine improves and facilitates the construction of operating system features like orthogonal persistence, object distribution, concurrency and capability-based security, giving a flexible integral OO computing environment.*

1 Introduction

The adoption of the object-oriented paradigm is not done in an integral way in all the system components. This produces a serious impedance-mismatch and interoperability problem, for the paradigm changes and/or object translations made depending on which element to work with. The result is a proliferation of additional software layers trying to alleviate these problems, but introducing in fact extra complexity in the system.

An approach in order to solve this problem is to move the OO support for the rest of the system to a common place into the OS. Oviedo3 [1] is a research project that tries to build an experimental integral object-oriented system based on that foundation. All components: user interfaces, applications, languages, compilers, databases... and the operating system itself share the same object-oriented paradigm.

The object-oriented operating system (OOOS) is the basis for this integral system. It provides only one abstraction: objects. Objects can only create new objects from a class or send messages to others. We believe an operating system should supply transparently the following features for the objects of this integral system: a capability-based protection mechanism, complete persistence, object distribution and concurrency. There are operating systems exploring these topics. The distributed OS Amoeba explores sparse capabilities in an object-based environment, and Mungi is a distributed single address space persistent OS with password capabilities. Another OS focusing on persistence is Grasshopper. Nevertheless, we think this particular mix of properties is very adequate for an integral OO system, and worth to research jointly in the context described below.

One technique to structure an OOOS aimed to support an integral OO system which offers many advantages is to use an OO abstract machine as the substrate of the OOOS. This machine offers the basic object model and support to all objects of the rest of the system. The OS functionality is given by a set of user objects not different from any other object.

2 The Object-Oriented Operating System

The heart of the Oviedo3 system is the operating system, named SO4 [1]. 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. In sections below, each one is analyzed in depth.

The master guidelines of the design of SO4 are:

- ? Object as the only abstraction. Only objects, of any granularity, exist.
- ? Intentionally "standard" object model, with the more common features found in OO programming languages: encapsulation, inheritance, and polymorphism [2].
- ? 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, achieving maximum simpleness.

Some important characteristics we wish to impose on objects, in accordance with these guidelines:

- ? Homogeneous objects. 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. Objects encapsulate all the information about it, including computation. The behavior must not be dependent on other objects.
- ? 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 is used as a reference is the only means to access it. The concept of memory address space does not exist.

3 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 abstract machine is shown. It has been designed specifically to support the OS as described above. This organization may be considered as a reference for the features that an abstract machine should have to support this kind of object-oriented OS. Other abstract machines could also be adapted for this purpose. Indeed, due to the popularity of the Java language, and the recent availability of public specifications we are considering the adaptation of the Java Virtual Machine [3].

The machine supports an object model with the following features: Object identity and abstraction, encapsulation, inheritance, and also generic and aggregation relationships between objects and, finally, polymorphism or message passing.

Figure 1 shows the architecture of the Carbayonia abstract machine ?1?, consisting of four areas: class area, references area, instance area, and system references area.

The class area maintains the description of each class. A set of primitive classes is permanently defined. The references area stores the references. Every reference has a type (relates to the class area) and points to an object of this type (relates to the instance area). The instance area stores objects created. And the system references area contains references with specific functions in the machine.

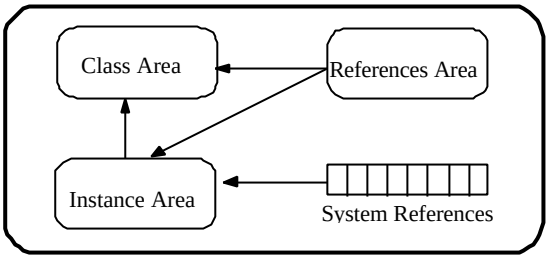

Fig. 1. Architecture of the Carbayonia abstract machine.

Each area can be considered as an object in charge of the management of its data.

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. 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:

- Impedance mismatch is removed. There is a common object model for the system, supplied by the machine for every object, including OS ones.
- Portability and 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. In sections below OS features are described, and how they benefit from the use of the abstract machine.

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 its inner workings 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. To allow OS objects to collaborate with the machine when it cannot accomplish a task by itself by means of a reflective

architecture[4]. The machine should be given an architecture where the machine objects are exposed as normal objects. When a machine object can not continue for some problem, it raises an exception, which 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 Security

Many operating systems use a protection mechanism based on the Lampson access matrix. Systems like Amoeba, Grasshopper, Mungi and Clouds use some kind of capability-based protection, while others like Guide, Amadeus, etc. use access control lists.

We intend to have a homogeneous access for every object based in two fundamental ideas:

- Capabilities [5] can be integrated as part of an object reference. They introduce a minimal overhead and the object model is not modified.
- Protection mechanism in the innermost level of the system, making the message passing mechanism check the protection information contained in the capability.

We propose to embody the protection management at the abstract machine level, converting references into capabilities, adding protection information to its contents.

New operations must be added to the machine, as well as modifying others to take into account the new role of references as capabilities.

Now, the machine must check the access rights after locating the object using its identifier in the capability. An exception is raised if no valid rights are held.

Thus advantages of both segregated and sparse capabilities [6] are combined. The machine guarantees capabilities can not be tampered with or altered without permission. Only the operations provided by the machine can be used to handle capabilities. Besides, neither the object model nor the way of using objects changes and the protection mechanism is transparent for the rest of the system. On the one hand, by using capabilities, the objects are not aware of the management of its protection. On the other, using capabilities as normal references in user structures facilitates system use, because there is no special way to access the segregated area where capabilities are stored.

6. Persistence

The function of the persistence system in SO4 is to provide a unique virtual persistent memory for objects, extending transparently the instance area by means of secondary storage. It creates a single persistent virtual space for objects. The main features desired are:

- Complete Orthogonal Persistence [7]. Every object is always persistent.
- Stability and Resilience [8]. The system must be able to resume operation after an unexpected system failure.

Some features of the abstract machine are very adequate to give support for persistence to the system:

- Unique object identifier. It is part of the references used in the objects. It eliminates pointer swizzling in disk swaps, as there is only one identifier for all situations.

? Self-contained Objects. A single operation stores the object on disk, including its computation, which is also encapsulated.

Putting together the above aspects a number of final advantages are achieved:

? Single memory abstraction. A single persistent virtual space replaces the duality of short-term and long-term memory, easier to understand and use for programmers and users.

? Continuous system and environment. The complete persistence of self-contained objects makes a truly continuous environment for every object, including OS ones.

? More intuitive interfaces. The system and the continuous environment permit interfaces closer to the user perception of the object functioning in the real world object. Disconnection, intended or not, does not alter their state.

? Virtual persistent distributed space. Together with the distribution system and using unique identifiers, a single distributed persistent object space among the machines of the system is created.

A way to support persistence is to use the exception ability of the machine to activate a “pager object” extending the instance area in a reflective way. Causes of activation may be object fault, object replacement, or arbitrary stabilization (figure 2).

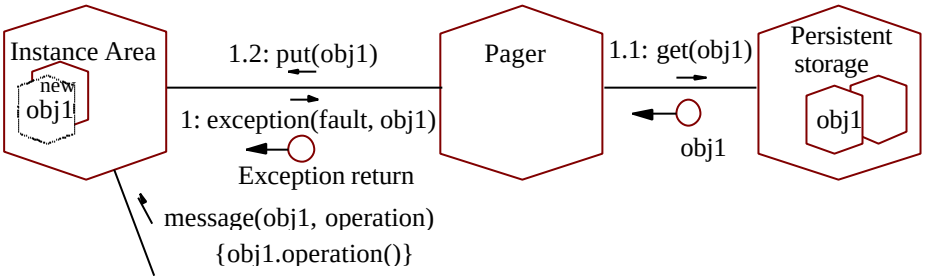

Fig. 2. Object Fault.

7 Distribution

On a distributed architecture consisting of several machines, the distribution system [9] allows inter-object communication without regard to its location, and provides mobility mechanisms with the maximum degree of transparency.

The goals of the distribution system [1] can be reduced to:

? Access and location transparency [10]. To provide a location service that delivers the messages to the objects, regardless of its location.

? Object mobility and load balancing.

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

? Unique object identifier. Objects are always referenced and accessed in the same way regardless of their location.

? Self-contained objects. Moving an object between machines is achieved just by moving its encapsulated state that includes the computation.

Combining these aspects some advantages are obtained:

? Single object distributed space.

? Performance improvement, by balancing the load of the whole system using object mobility.

? Reliability improvement, using a replication mechanism (as in Chorus or Clouds), possibly in combination with the persistence mechanism.

The use of a location object server as in the Clouds system and a load-balancing object is an approach to implement distribution.

Figure 3 shows a possible scenario of invoking a method in an object and the interaction between the persistence and distribution mechanisms.

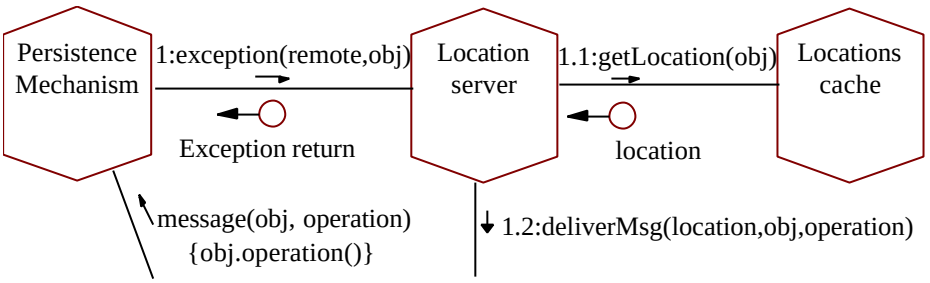

Fig. 3. Delivering a message for a remote object

A load-balancing object implements the object mobility policy of the system, based mainly on the load of every machine and the interaction with remote objects.

8 Concurrency

We try to endow the system with a *simple* concurrency model [1] that achieves the *maximization of the parallelism degree* in a secure way, allowing concurrency between objects and between methods of the same object, in the most secure way.

These are the main aspects of the concurrency model proposed for the system:

? Multithreaded active objects [9, 11]. Active objects that encapsulate computation are a natural extension of the self-contained object model provided by the abstract machine. Conceptually, they have a virtual multiprocessor for multiple threads, with the needed concurrency control mechanisms. Other existing OS such as Clouds and Guide chose a passive object model and supply an abstraction like process or thread to provide computation.

? Synchronous invocation [11]. For simpleness reasons, a thread of an object is blocked during the invocation of a method of another object.

? Exclusive and concurrent methods. When defining a method, it must be tagged as exclusive or concurrent. An exclusive method modifies the state of the object and cannot coexist with any other thread in the object. Objects behave as monitors when dealing with this kind of methods. Concurrent methods do not modify its state, so their threads are compatible with other concurrent threads. This approach has become very popular and recently languages such as Java have a similar concurrent model [12].

A set of objects provides the concurrency mechanism functionality. It seems more adequate to include some of this functionality in the machine and the rest in objects

providing OS functionality. Method execution ability and the characterization of methods as exclusive or concurrent is offered by the machine. OS objects will implement concurrency policies.

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

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. Thus the separation among base machine, operating system and user applications is removed: a single integral OO computing environment (figure 4).

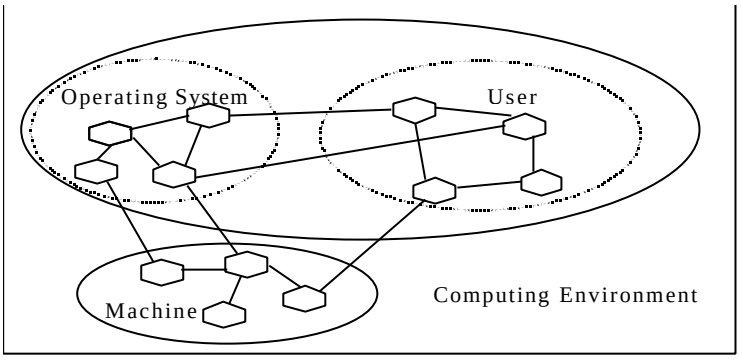

Fig. 4. Computing environment composed of a set of homogeneous objects

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

- Removal of non-necessary objects. Customized computing environments can be built as needed.
- Object replacing. It is possible to dynamically replace an object for another providing the same services in a different way.
- Reusability. Every object is organized into a class hierarchy. This allows code reuse, using the inheritance of the system. It is not necessary to re-implement functionality already present in other objects (even from the OS or the machine itself).

The combination of the three above elements produces a very flexible computing environment, easily adaptable and extensible, but in a secure way controlled by the machine capability protection mechanism.

There are other research projects with this goal of an integral OO system. Tunes [13] and Merlin [14] are systems which share this approach of using reflective architectures to obtain such environments, organized around the SELF language in the case of Merlin, and with a new language developed for Tunes.

10 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 OOOS.

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 OOOS.

The most relevant aspects of the OOOS are security, persistence, distribution and concurrency. The design decisions presented for every of these aspects, coupled with the OO abstract machine will give the user a computing environment completely based on the object oriented philosophy, more flexible, coherent, intuitive and easier of use, removing most of the problems of existing systems.

Oviedo3 is a research project at the University of Oviedo that intends to develop an experimental integral object system with this structure, using the Carbayonia OO abstract machine and the SO4 OO operating system.

A first version of the abstract machine (developed under Windows NT) is complete. The detailed design of the operating system components has now begun, for a subsequent implementation on the abstract machine.

References

[1] Cueva Lovelle, J.M., 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).

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

[3] Sun Microsystems Computer Corporation, "Java Virtual Machine Specification, Version 1.0 Beta", URL ftp://ftp.javasoft.com/docs/vmspec.ps.z, November 1996.

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

[5? Dennis, J.B., and E.C. Van Horn, "Programming Semantics for Multiprogrammed Computations", Communications of the ACM, Vol. 9, N.3, 1966.

[6? Anderson M., and C. Wallace, "Some comments on the implementation of capabilities", The Australian Computer Journal, 1988.

[7] Atkinson, M.P., P. Bailey, K. J. Chisholm, W.P. Cockshott, and R. Morrison, "An Approach to Persistent Programming", The Computer Journal, vol 26, 4, pp 360-365, 1983.

[8] Dearle, A., J. Rosenberg, F. Henskens, F. Vaughan, and K. Maciunas, "An Examination of Operating System Support for Persistent Object Systems", Proc. of the 25th Hawaii International Conference on System Sciences, Hawaii, U.S.A., pp 779-789, 1992.

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

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

[11] Papathomas, M., "Concurrency Issues in Object-Oriented Programming Languages", in Object-Oriented Development TR, Centre Universitaire d´Informatique, University of Geneva, ed. Tsichritzis, D., 1989.

[12] Sun Microsystems Computer Corporation, "The Java Language Specification", URL ftp://ftp.javasoft.com/docs/langspec-1.0.ps.Z, November 1996.

[13] The Tunes project, URL http://www.eleves.ens.fr:8080/home/rideau/Tunes/, November 1996.

[14] The Merlin Project, URL http://www.lsi.usp.br/~jecel/merlin.html, November 1996.