Object-Support Operating Systems

Position paper for the
Workshop on Operating Systems and Object Orientation
at ECOOP–OOPSLA 1990

Marc Shapiro
INRIA, BP 105, 78153 Rocquencourt Cédex, France
e-mail: shapiro@sor.inria.fr

July 1990

1 Introduction

The SOR group of INRIA (Systèmes à Objets Répartis—Distributed Object Systems) is engaged in the design and implementation of distributed object-support operating systems. We have built two prototypes: SOS and Cool. The former is generic and strives for application and language independence. The latter is more specific, being targeted at a particular language/application. Building on our experience with these two prototypes, we are starting a new project. Its goal is to define a general object-oriented framework for the design of distributed *object-support operating systems*; to implement a hierarchy of classes for object support; and to construct a family of specific subsystems based on the above framework and classes.

2 Operating systems and object orientation

Many OS projects claim to build an "object-oriented operating system", but there is no consensus to what this means. In my opinion there are at least three different uses of the term:

1. A process-and-message based OS, in which server processes encapsulate multiple, well-identified entities with clear interfaces. This category includes for instance Amoeba [9], as well as Spring and Vanguard (as far as I can tell from available information).

2. An OS designed using the object-oriented approach. Good examples are Choices [4] and the x-Kernel [8].

3. An OS with support for user-level objects, in the sense of separate entities with strong semantics. Typical of this category are SOS [14], Cool [7], and Guide [2]. Amber [5], and Hermes [3] also seem to be in this category.

An OS of category 1 would better be called a "server-oriented system", since the smallest executable entity it supports is the server.

A system of category 2 does deserve to be called object-oriented. However note that its object-orientation applies to its internal design, not necessarily to its interface. In fact, both Choices and the x-Kernel appear to provide rather traditional services to users, who remain unaware of their object orientation.

A system of the third kind is best categorized as an "object-support operating system"[1]. In contrast to the previous kind, such a system may very well have a traditional internal design, but it offers specific support mechanisms for objects, such as system-level identification of user-defined objects, finding objects, as well as migration, storage and garbage-collection of user-defined objects.

An OSOS differs from existing object-support runtime libraries, as it is not tied to any specific language or application area. An OSOS offers common object management, with the following consequences:

- a simpler universe for the development of applications;
- easier implementation of compilers and run-time environments for object-oriented languages;
- more efficient applications;
- communicating, and sharing, objects between independent applications.

Let us now examine two examples of OSOS: SOS and Cool.

3 The Object-Support Operating System SOS

SOS is distributed OSOS prototyped on top of Unix [12]. SOS supports an elementary object model which is both simple and powerful.

SOS is designed to encourage the use of the *proxy principle* [11] for structuring distributed applications. It extends the object concept to distributed, or "fragmented" objects. Externally, a fragmented object appears to be a single object. Its interface is provided by its local fragments or proxies, which are elementary objects. Internally, the many fragments are distributed.

A distributed service is implemented as a fragmented object. A client accesses the service by invoking a local fragment, called its *proxy* for the service. A client may acquire access to a new service by *migrating* in (e.g. via a proxy of the fragmented object manager) a proxy for that service.

Therefore a basic mechanism of SOS is object migration. A migrating object carries with it a list of *prerequisites*, i.e. objects which must be present in its environment; if necessary the system will automatically migrate (a proxy of) them also. A typical use of prerequisites is the code object, which carries type information, and methods to be linked dynamically[2].

An upcall interface allows objects to impose their own policy onto the system-defined migration mechanism. The victim is upcalled once before migration. The object's method for this call takes decisions such as migration by move or by copy, of itself or of some other object, with or without communication and/or replication of updates, etc. After migration, a finalization *reinitializer* upcall allows the object to re-establish its environment. This is used for instance to reset pointers[3] or

[1] *OSOS*—not *OS*/2 but indeed OS^2 !

[2] We have implemented a dynamic linker for C++ [6].

[3] We have implemented a context-independent pointer class permPtr [13].

request further importations. For instance, the `code` reinitializer performs the actual type-check and dynamic linking.

The mechanisms described above are language independent and generic. To support objects from a different language than C++, possibly with a different object model, it should suffice (hopefully) to attach appropriate prerequisites and upcall methods to those objects.

SOS is built using its own mechanisms: all the SOS system services are implemented as fragmented objects with local proxy interfaces. These include an object manager (called the Acquaintance Service), a flexible Name Service, and an object-oriented Communication Service.

4 The Object-Support Operating System Cool

Cool (the Chorus Object-Oriented Layer) extends the facilities of Chorus with support for object-oriented environments[4].

A Cool object is mobile, i.e. capable of migrating. It can be either passive (executes only within an invocation) or active (contains a thread). Cool objects are "medium-grained", meaning small enough that any number may coexist in any address space, but still too heavyweight to make every user-visible entity into a Cool object. A set of attributes, associated with an object, determines whether it is *globally* known and whether it is *persistent*. A global object may request receipt of messages sent by remote objects. A persistent object survives system shutdowns by causing its (current) address space of residence to be persistent.

The user part of an object consists of two segments:

- The code segment contains the code of the object methods. This segment is shared by all the instances of the same class, on the same machine.
- The data segment contains the object state. It constitutes the private "heap" of the object, and can grow or shrink following the object's dynamic (de)allocations. Its association with a specific "mapper" [1] or external pager allows to set its memory management policy.

Object migration is integrated with remote invocation. Any number of objects can migrate (by copy or by move) along a request or a reply.

A generic object-management kernel on top of the standard Chorus nucleus [10] implements functions such as object creation, deletion, storage, remote invocation and migration. This kernel is part of the Cool "sub-system", i.e. a collection of servers, some executing in supervisor mode, which collectively implement a protected system-call interface.

We can implement group operations, such as migration of several objects, more efficiently in the kernel than in external servers. Multiple object models may be supported by run-time environments above this kernel; currently we only have a C++ run-time.

5 Upcoming work

We are currently defining (together with Chorus-Systèmes) a new project called Soul, which will be both an object-oriented OS and an OSOS.

[4]This section is adapted, with permission, from [7].

Soul is intended to combine the best features of SOS (e.g. fragmented objects) and Cool (e.g. clean layering and efficiency). The kernel has similar functions to the existing Chorus nucleus, extended to allow the grouping of arbitrary resources, which can then be acted upon (e.g. identified, copied, or migrated) together.

Right above the kernel, we envisage a hierarchy of object-support object types and classes. These can be re-used, parameterized, and combined together, in order to build specific object-support functions for the upper layer.

The latter is a collection of subsystems, i.e. specialized virtual-machine interfaces. For instance we plan to build an SOS subsystem and an Amber-like subsystem, each with its own system calls, re-using classes from the hierarchy.

References

[1] V. Abrossimov, M. Rozier, and M. Shapiro. Generic virtual memory management for operating system kernels. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pages 123–136, Litchfield Park AZ (USA), December 1989. ACM.

[2] R. Balter, S. Krakowiak, M. Meysembourg, C. Roisin, X. Rousset de Pina, R. Scioville, and G. Vandôme. Principes de conception du système d'exploitation réparti Guide. Rapport Guide R1, Laboratoire de Génie Informatique, Saint-Martin-d'Hères (France), April 1987.

[3] Andrew P. Black and Yeshayahu Artsy. Implementing location independant invocation. In *Proceedings of the 9th Int. Conf. on Distributed Computing Systems*, pages 550–559, Newport Beach, CA USA, June 1989. IEEE.

[4] Roy H. Campbell, Gary M. Johnston, Peter W. Madany, and Vincent F. Russo. Principles of object-oriented operating system design. Technical Report R-89-1510, Department of Computer Science, University of Illinois, Urbana, Illinois (USA), April 1989.

[5] Jeffrey S. Chase, Franz G. Amador, Edward D. Lazowska, Henry M. Levy, and Richard J. Littlefield. The Amber system: Parallel programming on a network of multiprocessors. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pages 147–158, Litchfield Park, Arizona USA, December 1989. ACM.

[6] Philippe Gautron and Marc Shapiro. Two extensions to C++: A dynamic link editor and inner data. In *Proceeding and additional papers, C++ Workshop*, Berkeley, CA (USA), November 1987. USENIX.

[7] Sabine Habert, Laurence Mosseri, and Vadim Abrossimov. COOL: Kernel support for object-oriented environments. In *ECOOP/OOPSLA'90 Conference*, volume 25 of *SIGPLAN Notices*, pages 269–277, Ottawa (Canada), October 1990. ACM.

[8] Norman C. Hutchinson, Larry L. Peterson, Mark B. Abbott, and Sean O'Malley. RPC in the x-Kernel: evaluating new design techniques. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pages 91–101, Litchfield Park, Arizona USA, December 1989. ACM.

[9] Sape J. Mullender and Guido van Rossum. Amoeba — high-performance distributed computing. Technical Report CS-R8937, Centre for Mathematics and Computer Science, Amsterdam (The Netherlands), 1989.

[10] M. Rozier, V. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrmann, C. Kaiser, S. Langlois, P. Léonard, and W. Neuhauser. Chorus distributed operating systems. *Computing Systems*, 1(4):305–367, 1988.

[11] Marc Shapiro. Structure and encapsulation in distributed systems: the Proxy Principle. In *The 6th International Conference on Distributed Computer Systems*, pages 198–204, Cambridge, Mass. (USA), May 1986. IEEE.

[12] Marc Shapiro. Prototyping a distributed object-oriented OS on Unix. In Eugene Spafford, editor, *Workshop on Experiences with Building Distributed and Multiprocessor Systems*, Ft. Lauderdale FL (USA), October 1989. USENIX. Also available as Rapport de Recherche INRIA no. 1082.

[13] Marc Shapiro, Philippe Gautron, and Laurence Mosseri. Persistence and migration for C++ objects. In *ECOOP'89*, Nottingham (GB), July 1989.

[14] Marc Shapiro, Yvon Gourhant, Sabine Habert, Laurence Mosseri, Michel Ruffin, and Céline Valot. SOS: An object-oriented operating system — assessment and perspectives. *Computing Systems*, 2(4):287–338, December 1989.