Fragmented Object: a Building Block for Distributed Object-Support Operating Systems

Yvon Gourhant and Mesaac Mounchili Makpangou

INRIA, B.P. 105, 78153 Le Chesnay Cédex, France

1 Introduction

The object oriented programming paradigm is increasingly recognized to be of primary importance in structuring large softwares. Remote invocations, object migration [5, 8], and shared objects [9, 2] have been proposed to facilitate the use of this paradigm for distributed software.

However, these works focus on the case of *elementary objects* (simple objects with local representation) that ought to be accessible from different address spaces. They do not adequately address the case of distributed services which are implemented by cooperative server objects, or by a server exporting proxies to the clients [14], or by several objects encapsulating the manipulation of a protected resource.

In all of these cases, several elementary objects instantiated in different address spaces cooperate closely with one another without interference from other objects. ¿From the outside world, these objects are seen as a single object: the location of the elementary objects (or *fragments*) constituting the *fragmented object* is transparent to the users. However, the implementor of a *fragmented object* is conscious of the distribution of these fragments. He can then benefit from this awareness by selecting the communication mechanisms which fit best its needs.

These cases are well addressed using the high-level, structured approach, called fragmented objects. The SOR (Systèmes à Objets Répartis – Distributed Object-Oriented Systems) group at INRIA has developed a distributed object-oriented operating system, called SOS, based on the fragmented object concept [15]. The fragmented object concept is used in SOS to structure a distributed object manager [7], communication protocols [12, 13], and a distributed name service [10]. This concept is beneficially used for structuring distributed applications as well; (e.g. BFIR2, a multimedia document toolbox, and Images, an user-interface management system [1]).

This paper aims to present this concept and to briefly discuss its implementation. The rest of the paper is organized as follows. In section 2 we define the concept. We then discuss in section 3 the language support and some communication facilities for its implementation. Section 4 compares this concept with some extensions of the object paradigm for distributed environments. Finally, section 5 concludes this paper.

2 Definition

A fragmented object (FO) is a shared distributed object. Its clients can be localized in several address spaces, possibly on different sites. It offers for each client, a strongly-typed programmer-interface (called a *client interface*) and a programmer-defined *binding procedure* (to bind the client to a specific client interface of the FO). The client interface provides to the client the full distribution transparency and hides its fragmented representation.

Internally, an FO is implemented as a group of elementary objects, its fragments. Each fragment is mapped onto one of the address spaces spanned by the FO. They cooperate using low-level FOs, such as communication channels. The communication between fragments is strongly typed-checked.

3 Implementation

To implement this concept, we define a new language called FOG (Fragmented Object Generator) [6] and a toolkit of low-level FOs implementing untyped-communications between address spaces in SOS. This section briefly presents the main features of the FOG language, the toolkit of low-level FOs and the FOG compiler.

3.1 Language support

3.1.1 Class group

Just as a simple object, a FO has a type. The type of a FO is specified by a *class group*. A *class group* is a composition of related fragment classes. The FOG language provides features for specifying this composition.

3.1.2 Fragment interfaces

A class of a fragment is composed of three parts:

- A public interface.
  The public interface of a fragment is a strongly-typed interface accessible by objects within the same address space. It makes up for the client interface of the FO in this address space.
- A private representation, accessible only within the fragment.
  It contains some code and data (like traditional object representation), plus some low-level FOs shared with other fragments.
- A *group interface*, only accessible by the fragments within the FO.
  It encapsulates access to low-level FOs and provides a strongly typed-checked interface for communication between fragments.

3.2 Toolkit of low-level fragmented objects

To assist the FO programmer, we provide a toolkit of low-level FOs that implements communication between fragments of higher level FOs. We distinguish two types of such low-levels FOs:

- FOs implementing remote invocation of fragments, e.g. communication channels implementing remote procedure call, asynchronous remote procedure call, parallel remote procedure call, ...
- FOs allowing fragments to share data (i.e. implementing replication and caching policies).

Currently, our toolkit contains only communication channels.

3.3 Compiler support

The FOG language is an extension of C++. Its compiler is a preprocessor to the C++ compiler.

The FOG compiler checks the consistency of the programmer declarations of group and client interfaces. It also checks the private use of shared FOs between fragments.

It automatically generates coercions between the group interface of an englobing FO and the client interface of a shared englobed FO: For a FO implementing remote procedure call, the FOG compiler generates the code for marshaling/unmarshaling invocation parameters into/from a message and handling remote exceptions. In addition, for an asynchronous and/or parallel remote procedure call, it generates a result handler that encapsulates the differed and/or successive result(s). The caller will then query the handler to obtain available results.

4 Related work

Among the many extensions of the object paradigm for distributed environments, the Gothic own model of FO [4, 11] is the closest to our concept.

The Gothic FO model is language-oriented. It is based on the multi-function paradigm (N callers and P callees) [3]. The multi-function paradigm permits different synchronization types: 1 to N (multicast-like semantic), N to 1 (rendez-vous), or N to P (rendez-vous followed by a multicast). All these synchronization mechanisms facilitates the programming of parallel programs using the Gothic FO model. Furthermore, all the fragments of a Gothic FO implement the same interface. Any invocation to a Gothic FO requires the participation of all its fragments. Finally, the implementor of a Gothic FO has no control of the communication protocol used by fragments.

Our model differs from theirs essentially in two points. First, we focus on distributed computations rather than on parallel ones. Second, we favor the system approach; especially, the SOS FO implementor has the ability to select the communication mechanisms that best fit its needs.

5 Conclusion

The main strength of the FO concept is that it provides the appropriate level of distribution visibility to the implementor, while hiding the distribution of fragments from the users. This two level transparency approach fits well to the respective needs of both implementors and users of most distributed system services and applications. From our experience with SOS, we feel strongly that this two level transparency approach allows the FO concept to server as a building block for both distributed object-support operating systems and applications.

References

[1] J. Alves Marques, J. Paulo Cunha, and L. Pinto Simoes. SOMIW user interface management system. (Notes of the SOMIW MTC meeting, June 25–27, Lisboa, Portugal), July 1986.

[2] H. E. Bal and A. S. Tanenbaum. Orca: A language for distributed object-based programming. *SIGPLAN Notices*, 25(5):17–24, may 1990.

[3] Jean-Pierre Banâtre, Michel Banâtre, and Florimond Ployette. The concept of multi-function: a general structuring tool for distributed operating system. In *The 6th International Conference on Distributed Computer Systems*, pages 478–485, Cambridge, Mass. (USA), May 1986. IEEE.

[4] Jean-Pierre Banâtre, Michel Banâtre, and Florimond Ployette. An overview of the Gothic distributed operating system. Rapport de recherche 504, INRIA, March 1986.

[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] Yvon Gourhant and Marc Shapiro. FOG/C++: a fragmented-object generator. In *C++ Conference*, pages 63–74, San Francisco, CA (USA), April 1990. Usenix.

[7] Sabine Habert. *Gestion d'objets et migration dans les systèmes répartis*. PhD thesis, Université Paris-6 Pierre-et-Marie-Curie, Paris (France), December 1989.

[8] Norman C. Hutchinson. Emerald: An object–based language for distributed programming. Technical Report 87-01-01, Department of Computer Science, University of Washington, Seattle, WA (USA), January 1987.

[9] S. Krakowiak, M. Meysembourg, M. Riveill, and C. Roisin. Modèle d'objets et langage du système guide. Rapport Guide R2, Laboratoire de Génie Informatique, Saint-Martin-d'Hères (France), November 1987.

[10] Jean-Pierre Le Narzul and Marc Shapiro. Un service de nommage pour un système répartis à objets. In *Séminaire Franco-Brésilien sur les Systèmes Informatiques Répartis*, pages 127–133, Florianopolis (Breésil, September 1989. LAAS (Toulouse), l'UFSC.

[11] Philippe Lecler. *Une approche de la programmation des systèmes distribués fondée sur la fragmentation des données et des calculs, et sa mise en œuvre dans le système Gothic*. Thèse de doctorat, Université de Rennes I, Rennes (France), September 1989.

[12] Mesaac Makpangou and Marc Shapiro. The SOS object-oriented communication service. In *Proc. 9th Int. Conf. on Computer Communication*, Tel Aviv (Israel), October–November 1988.

[13] Mesaac Mounchili Makpangou. *Protocoles de communication et programmation par objets : l'exemple de SOS*. PhD thesis, Université Paris VI, Paris (France), February 1989.

[14] 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.

[15] 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.