Distributed Programming in GArF *

Benoît Garbinato, Rachid Guerraoui, Karim R. Mazouni
Département d'Informatique
Ecole Polytechnique Fédérale de Lausanne
CH-1015 Lausanne, Suisse

Abstract. GArF is an object-oriented programming environment aimed to support the design of reliable distributed applications. Its computational model is based on two programming levels: the functional level and the behavioral level. At the functional level, software functionalities are described using passive objects, named data objects, in a centralized, volatile, and failure free environment. At the behavioral level, data objects are dynamically bound to encapsulators and mailers which support distribution, concurrency, persistence and fault tolerance. Encapsulators wrap data objects by controlling how the latter send and receive messages, while mailers perform communications between encapsulators. This paper describes how the GArF computational model enables to build flexible and highly modular abstractions for the design of reliable distributed applications.

1 Introduction

Transparency has been recognized to be a worthwhile goal in the design of distributed systems. A user of a transparently distributed system could make no distinction between local resource accesses and remote ones. From the point of view of a distributed application designer, we believe that transparency has to be provided "à la carte". That is, a distributed programming environment must provide programmers with a set of behavioral abstractions that hides low level features related to concurrency, persistence, distribution and fault-tolerance. Nevertheless, programmers must be able to refine such abstractions in order to obtain the desired efficiency. For example, the underlying system may decide to locate, by default, the application's components on different nodes, according to a particular load balancing policy. However, the actual distribution of the program is sometimes likely to be closely associated with the semantics of the program. Therefore, programmers must be able to explicitly locate components. There are many other examples related to concurrency and replication control where default behaviors are too restrictive and where it is often desirable to use application's semantics to refine these behaviors.

In this paper, we describe the computational model of GArF: an object-oriented distributed system that supports transparency "à la carte". GArF provides built-in behavioral abstractions (i.e., abstractions that deal with concurrency, persistence, distribution, and fault-tolerance), which can be used as such, or refined at programmers' convenience.

GArF promotes software modularity by separating the functional aspects of applications, from their behavioral features. The former are designed within data objects, similar to objects

* Research supported by the Swiss FNS as part of the project Mesures Spéciales under contract number 5003-034344.
in classical centralized languages, whereas the latter are confined within particular objects named *encapsulators* and *mailers*. *Encapsulators* are used to control how data objects send and receive messages, whereas *mailers* are used to perform communications between encapsulators. GARF provides a built-in extensible library of encapsulator and mailer classes that implement very flexible behavioral features such as client/server asynchrony, active/passive replication, semantic-based concurrency control, etc. These classes can be used as such, or refined according to application's semantics.

The remainder of this paper is organized as follows. Section 2 describes the GARF computational model. Section 3 shows its flexibility and modularity through some distribution related examples. Section 4 presents a simple distributed and fault-tolerant application implemented in GARF. Section 5 contrasts GARF with related work and section 6 concludes with some general remarks.

## 2 The GARF computational model

### 2.1 Two programming levels

The GARF computational model is based on two programming levels. A *high* level, called *functional level*, in which the programmer describes aspects of the application that could be expressed in a centralized, volatile, and sequential object-oriented system; and a *low* level, called *behavioral level*, in which programmer describes behavioral features related to concurrency, persistence, distribution, and fault-tolerance.

**The functional level:** at this level, the programmer focuses on all sequential aspects of the application, assuming a volatile, centralized, and failure-free environment. Such (functional) aspects are described through classes, called *data classes*, of which instances are called *data objects*. The programmer can here reuse classes designed within a classical object-oriented language (Smalltalk in our current prototype).

![Fig. 1. Two programming levels](image)

**The behavioral level:** at this level, programmer deals with features related to concurrency, persistence, distribution, and fault-tolerance. Such (behavioral) features are implemented within classes, called *behavioral classes*, of which instances are called *behavioral objects*. GARF provides two kinds of behavioral classes: *encapsulator classes* of which instances