Definition of a Reflective Kernel for a Prototype-Based Language

Philippe Mulet, Pierre Cointe
Ecole des Mines de Nantes
3, rue Marcel Sembat
44049 Nantes Cedex 04
(+33) 40.44.83.87 - Fax (+33) 40.71.97.40
[mulet,cointe]@info.emn.fr

Abstract
We present the implementation of Moostrap, a reflective prototype-based language, the interpreter of which is written in Scheme. Moostrap is based on a reduced number of primitives, according to a previous work for defining a taxonomy for prototype-based languages.

Our purpose is to reify the behavior of any object through two steps: the slot lookup and its application. The first phase is reified thanks to behavioral meta-objects, and the second is managed by special objects, called slot-executants.

This kernel does not handle any implicit delegation at first. However, we introduce it, as a first extension of the basic language using a new behavioral meta-object.

Keywords: Prototype, Reflection, Primitive, Slot, Creation, Cloning, Delegation, Extensibility, Behavior, Meta-Object, Self, Smalltalk, Scheme.

1 Introduction

Our general area of research is the study of reflection in object-oriented languages (OOLs). We have previously defined a model of behavioral reflection expressing the computation of a message as the combination of a message lookup and a message application.

We have implemented and experimented this model in a newly defined prototype-based language, which bootstraps itself from a reduced set of primitives, and named Moostrap (standing for “Mini Object Oriented System Towards Reflective Architectures for Programming”). The purpose of this paper is to describe this reflective language.

Moostrap belongs to the family of prototype-based languages, a basic alternative to class-based languages. Indeed, we decided to free our language from classes, because they hardly deal with one-of-a-kind objects. Furthermore, we believe that classes interfere with the “emerging” notion of meta-objects [MDC92].

Initially, we realized a Smalltalk’80 implementation of the Self 1.0 prototype-based language [MR91], that we then extended to perform reflective computation [CMDM92].
Our purpose was also to add reflective facilities to control the computation rules of message sending.

In standard Self, prototype structure can be consulted thanks to mirror objects [CUL89]. These first-class objects are created explicitly to provide an interface between a prototype and its internal representation. Unfortunately, mirrors are not causally connected to the internal structure they only give a reading access to and are mainly used to create virtual slot-descriptor objects [MCD91]. In fact, mirrors are not supposed to play the role of behavioral meta-objects; we rather decided to introduce explicit meta-objects (to handle the lookup) and a new reification mechanism for slots (to handle the application process and to allow slot extension). This approach allowed us to roughly validate our scheme for behavioral reflection in the context of Self. However our explicit reification of slots was not nicely integrated to Self, and rather than changing the kernel of Self (and its virtual machine) we decided to design our own language and to implement its first interpreter in Scheme [Sch89].

This paper is divided into four parts. We first describe Moostrap as a prototype-based language, then we introduce the reflective model of this language. In a third part, we extend the kernel, thanks to reflective facilities, in order to add implicit delegation, a kind of inheritance between prototypes. Then we present some implementation details and conclude on our future work.

2 Moostrap: a Tiny Prototype-Based Language

The taxonomy of prototype-based languages, established in [DMC92], assumes that these languages are basically built on objects communicating by message sending and supporting cloning facilities. The classification is then made according to the following criteria: representation of information either unified as slots or separated as variables and methods, dynamic modification of object structures, ex-nihilo objects creation, explicit or implicit delegation, and handling of split-objects due to delegation.

According to this taxonomy, Moostrap is classified as a language where the prototypes store information as slots. It allows dynamic modification of prototype structures, creation of objects ex-nihilo empty or even with an initialized structure, but does not provide any mechanism of delegation. This is however supplied by implicit sharing between prototypes achieved by a primitive for adding shared slots.

2.1 Components of the Language

The set of primitives of Moostrap prototype-based language is deliberately kept small. Ex-nihilo objects can be created thanks to a special syntax; we only regard as primitives the following operations: cloning objects, adding slots to an object or removing them.

2.1.1 Ex-Nihilo Declaration of Prototypes

Prototypes are made up of slots. In Moostrap, the notion of slot corresponds to an association \((<\text{name}> <\text{value}>),\) a type of dynamic binding. We distinguish two kinds of