OOPSLA '93 Workshop on Reflection
and Metalevel Architectures in Object-Oriented Programming

Browsing in Explicit Metaclass Languages : An Essay in Reflective Programming Environments

François Rousseau, Jacques Malenfant
Université de Montréal
C.P. 6128, Succursale A
Montréal, Canada, H3C 3J7
e-mail: {rousseaf, malenfan}@iro.umontral.ca

Abstract

The development of adapted programming environments is mandatory in order to deliver reflective languages to real world users. It is generally assumed that reflection aids in the development of programming environments. However, its use may also give rise to certain difficulties. For example, it prevents programming tools from making strong assumptions about language structure and semantic. Our paper tackles this rather large problem by concentrating on the case of the browser in the context of explicit metaclasses. We first propose a new browser architecture that is better adapted to explicit metaclass languages. The new browser uses a decentralized approach to manage information related to the internal structure of classes. The decentralized approach involves the use of third-party objects named Descriptors, which provide individual class representations. Next we implement this architecture in Classtalk [2,8] under VisualWorks [6]. The VisualWorks interface generator is used to assist metaclass designers with the task of incorporating descriptors for new class structures into the browser. This latter is effective on several ClassTalk metaclasses as, for example, on a CombinationClass, which has three method dictionaries.

1 Introduction

A widely accepted view among the reflexive community is that a language's reflective features should facilitate the implementation of the programming environment. However little has been said about the adaptation of programming environments around such reflective features. The browser is one of those programming tools sensible to reflective features. In this paper, we describe a number of modifications that improves its adaptability as a programming tool for explicit metaclass languages. First we review the major limitations of current browsers in relation to the new needs. Next, we describe approaches for a new browser that may eliminate most of the adaptability problems programmers are confronted with. Finally using ClassTalk, under VisualWorks, we present an implementation of the design scheme we consider best suited to meet the requirements.

2 Browsing in explicit metaclass languages

2.1 Explicit metaclass languages

Explicit metaclass languages [1,8,9], as ObjVLisp, allow us to extend the language through development of heterogeneous class structures. Traditionnally, classes had only one method dictionary. By contrast, using an explicit metaclass language offers the possibility of defining a new metaclass, for example CombinationClass [2], whose instances will implement method combinations with before-primary-after methods through three method dictionaries.

2.2 The browser adaptation problem

The browser is a graphic application allowing edition of the system classes content. This edition implies representation, manipulation, and management of the class components. With respect to explicit metaclass languages, the standard browser presents the problem of being unable to access information directly related to each individual metaclass instance. Moreover, means to manage existing heterogeneous class representations and future introductions must be found. Which criteria should the browser design be based on ? In addition to the basic functionalities expected from a standard browser, there is a list of characteristics we would consider desirable as well:

1. Users require a browser that can represent each and every class as done in implicit metaclass environments e.g. Smalltalk-80 [7].
2. Any complex information introduced by explicit metaclasses should be hidden by the interface.
3. It should be relatively easy for the metaclass designer to add, in the browser, concepts associated with his new metaclasses.

3 A browser for an explicit metaclass language

3.1 Centralized versus decentralized approaches

A simple solution to the adaptation problem is to include in the browser all functionalities required to support the different structures found in the system. However, this approach has three major drawbacks:

1. Since the browser would be highly complex and lacking modularity, this approach leaves little room for future improvements.
2. Such a browser would be making strong assumptions on the classes' nature, and this is an inconvenience in a reflective environment.
3. Finally, the browser would be subject to modification imposed by any metaclass designers involved in the system development.

An alternative approach, here on reffered as the *decentralized approach*, avoids all the problems generated by the multiplicity of structures involved. With this approach, the browser does not make any assumptions because all the information related to the representation and manipulation of class structures has been removed. The browser retains only those functions that enables it to navigate through the classes and to access complementary information required for class representation and manipulation. In other words, the browser keeps only those functions that pertain to system organization, not to class structures. The decentralized approach is covered with greater depth in the remaining sections.

3.2 A decentralized approach using metaclasses

As previously mentioned, the decentralized approach physically takes away from the browser a certain type of information. To this point, however nothing has been said about the places of storage and means to access the displaced information. There are in fact two alternatives: one can use 1) metaclasses, or 2) third-party objects, named Descriptors. We consider advantageous to place edition-related information in metaclasses for the following reasons:

1. The metaclass whose instances are being edited are easily accessible through the instantiation link.
2. Information about the structure and behavior of their respective instances already exists in each metaclasses.

3. Inheritance between metaclasses allows us to share representation and manipulation information between inherited structures.

There are nonetheless certain disadvantages associated with the metaclass approach:

1. Class editing information has to implicitly follow the inheritance link due to its location within the metaclasses. Hence, it cannot use a separate hierarchy specifically designed for graphic representation. For example some classes could share chracteristics with intermediate or abstract classes.
2. Metaclasses are already overloaded with multiple uses. This approach would place even more demands on them.

Because of these disadvantages, we favor the use of third-party objects or Descriptors. This alternative approach allows us to keep information presented to users separate from that related to the representation and manipulation of classes. Descriptors are accessible to metaclasses and, therefore, also accessible to the browser. The way descriptors is made accessible to the browser is presented in figure 1.

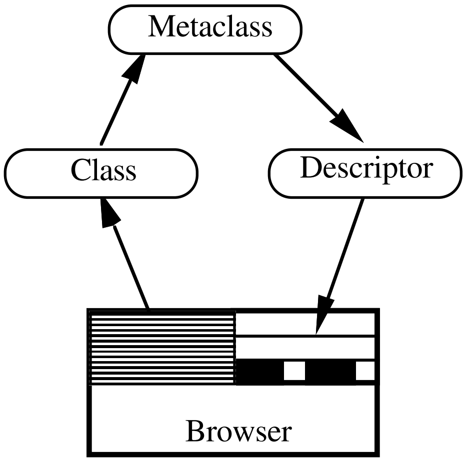

Figure 1. Links between the browser and the descriptors

3.3 Functional description of the proposed model with Descriptors

The use of descriptors can be depicted by a model as the one presented in figure 2. In this model, the browser maintains its ability to navigate throughout all classes. Descriptors, as objects, must perform two essential tasks:

1. draw all of its graphic components inside the browser interface,
2. support manipulations of these graphic components.

For example, on selection of a class A, the browser requests from the metaclass MA that it takes care of the edition of A. The metaclass MA in turn sends this request to its associated descriptor DA. This last class, DA, starts an instance inside the browser, which takes care of the representation of A and responds to user manipulations. Modifications brought to class A are managed by the DA instance, the latter insures that edition is properly done and stored. A blank zone inside the browser is saved exclusively for storing the display of the descriptor interface. Upon selection of another class, B, the interface display stored in this zone is replaced by a new interface corresponding to the class B.

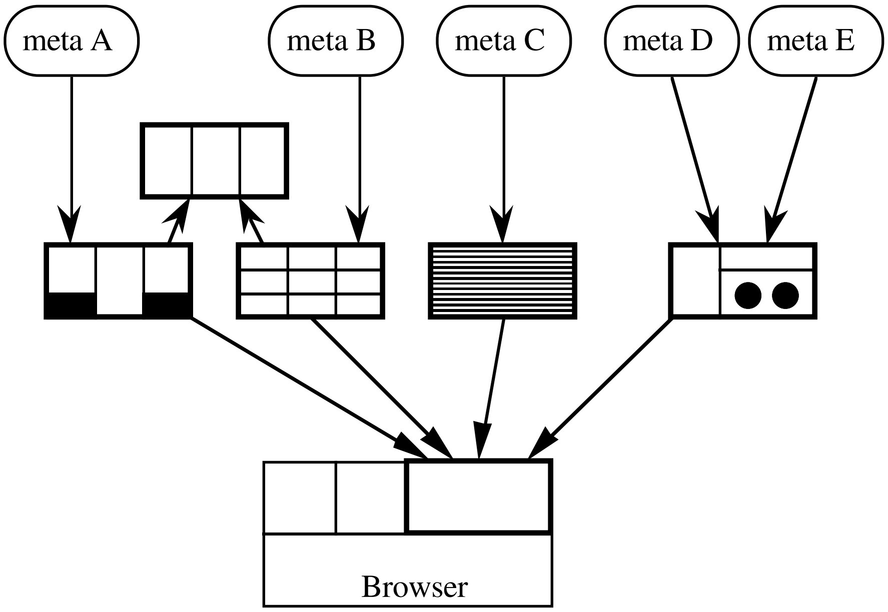

Figure 2. Descriptors are assigned a specific zone into the browser

The browser is devised to receive descriptors with a specific format. All descriptors in the system are designed to complement the browser. These are conceived to fit in the browser interface from where they can answer all standard editing commands such as text edition, action on menus, list selection, and so on. The metaclass designer must provide a specific descriptor (if required for a new structure) as well as all the information required for representation and manipulation of its new model.

These tasks should be kept as simple as possible. Keeping this in mind, the use of an interface generator may prove extremely helpful:

1. firstly because it doesn't require understanding of the way system graphics are implemented, and
2. secondly because it usually generates all its applications in a unique format, most if it inherited from a base class. Uniformity greatly facilitates the integration of the descriptor interface into that of the browser.

Once a skeleton has been obtained from the interface generator, the designer must provide it with various methods assigned to the management of classes. This step requires knowledge about the nature of metaclasses and format produced by the interface generator. Since descriptor creation is, to a large extent, accomplished through hierarchical inheritance, the designer task is simplified to making small changes on the inherited descriptor.

4 ClassTalk under VisualWorks

4.1 Description of ClassTalk and VisualWorks

ClassTalk integrates the ObjVLisp model inside Smalltalk-80 [8]. Some ClassTalk metaclasses impose a different class structure, which the current Smalltalk browser cannot recognize and thus represent. The new browser that we have created using the VisualWorks interface generator does recognize these different class structures. VisualWorks is also used to produce descriptors required by certain metaclasses.

4.2 Implementation of the proposed functional description

The first implementation step consisted of reconstructing the browser so that it could integrate descriptors. The browser contains a list of categories, a list of classes, a blank zone occupied by descriptors, and a text edition zone (see figure 3 below).

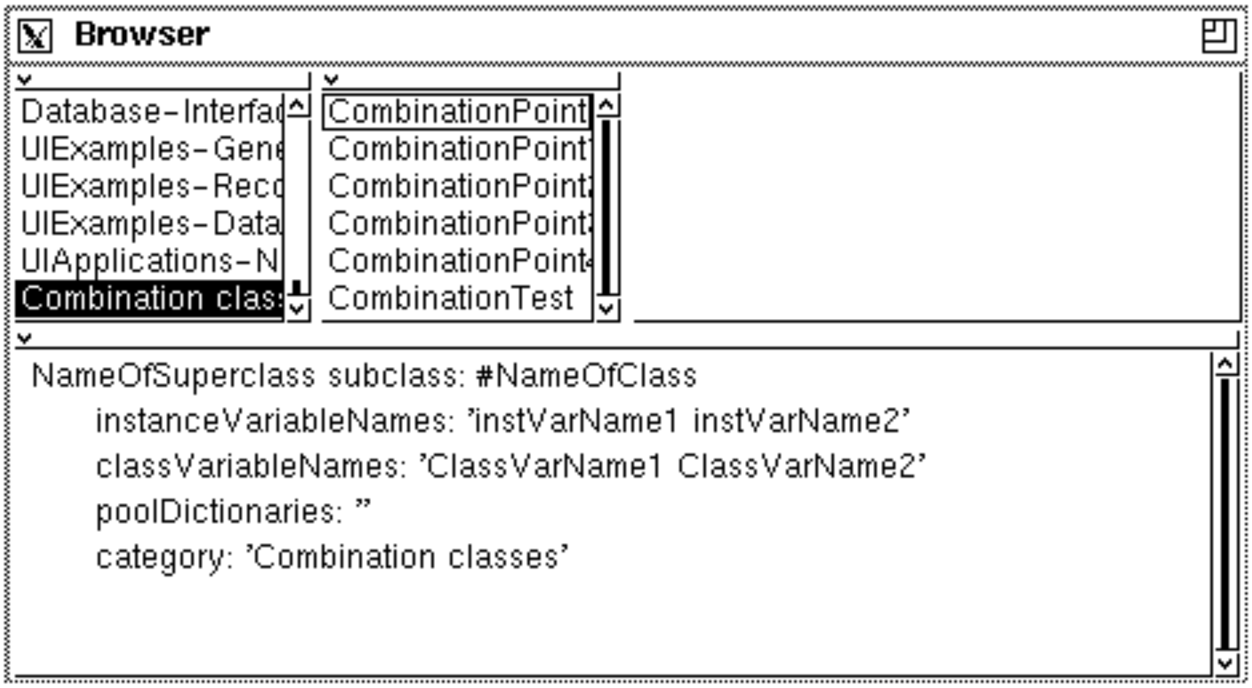

Figure 3. Our browser without a selected class.

The blank zone in its current form is fixed with respect of its use and format and it occupies the space normally reserved to protocol and method lists. The text edition zone is accessible to the descriptor but strictly for editing purposes. The user can navigate through categories to select a specific class. Menu options for categories and classes are directly implemented in the browser level.

Upon selection of a specific class, the following method is executed:

```
editClass
      self selectedClass openEditorOn: (self selectedClass) viewer: self
```

This method triggers the following sequence of events:

First, the metaclass is accessed in order to open the editor (the descriptor). The class selected and a pointer on the browser (self) are two arguments being passed on the descriptor. This gives the descriptor free access to text edition zone. Figure 4 illustrates a situation where the user selects a method combination class.

The described browser does not need to be modified if any new structures are introduced into the system. However, the designer must make a descriptor available to the browser every time a new metaclass introduces a structure. This ensures that the new metaclass instances are accessible for edition.

4.3 Directions for the metaclass designer

Since a descriptor is itself a class that inherits from already existing descriptors, designers do not have to start from scratch. The design include:

1. Creation of a graphic interface with the help of VisualWorks. The output from VisualWorks is class material that may be considered as a descriptor skeleton. This class inherits either directly from another descriptor or from a VisualWorks base class (ApplicationModel).
2. Incorporation of action methods associated with the interface graphic conponents. These methods use other methods located in the metaclass to perform final modifications to the class components. For example, the descriptor must send the message: "getAllProtocolsOf:theClass" to the class being edited so its protocol list is filled correctly. This last method must be implemented in the metaclass of the class being edited (theClass).
3. Addition of methods describing class structure elements to the metaclass. In the previous example the method "getAllProtocolsOf:" must be made available at the metaclass level.

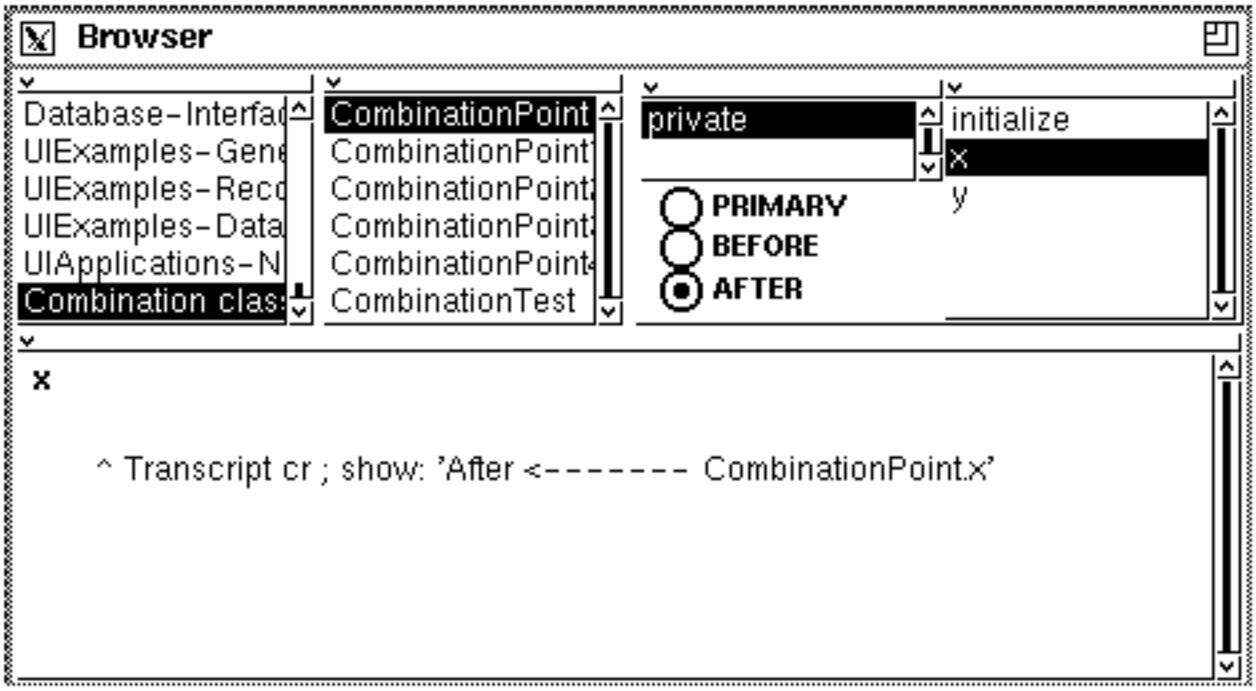

Figure 4. Our browser opened on the method combination class CombinationPoint

5 Conclusions and perspectives

Browser limitations in programming environments with explicit metaclasses may be solved by removing from the browser all class editing tasks.The proposed browser gives access to all classes but requires third party objects, or *Descriptors* to manage the specific class representation and manipulation. Methods stored in metaclasses are used by other methods located in associated descriptors to access class information. The metaclass designer must deliver a dedicated descriptor with each special metaclass so that edition of their instances can proceed in the browser. The proposed model was implemented by using ClassTalk for the design of all metaclasses, and the VisualWorks interface generator for all visual aspects stored in descriptors.

Objectives for the near future include new design tools to facilitate the task of metaclass designers. These tools should abridge the relatively long process of bringing refinements to the crude descriptor skeleton produced by the interface generator. Our current browser does not offer the class spawning option normally included with browsers. We will see to the design and implementation of this option so as to reduce the burden on the designer. Finally, we plan to remove some of the rigidity associated with the sharing of the text edition zone. A more flexible text edition zone could be put to other uses and tailored to the needs of specific class structures.

References

[1] P. Cointe. Metaclasses are First Class: the ObjVlisp Model. Proc. of OOPSLA'87, ACM Sigplan Notices, 22(12):156-167, December 1987.

[2] P. Cointe. The Classtalk System: a Laboratory to Study Reflection in Smalltalk. In [3].

[3] Proc. of the First Workshop on Reflection and Metalevel Architectures in Object-Oriented Programming, OOPSLA'90, October 1990.

[4] Proc. of the Second Workshop on Reflection and Metalevel Architectures in Object-Oriented Programming, OOPSLA'91, October 1991.

[5] A. Yonezawa and B. Smith, editors. Proc. of the International Workshop on New Models for Software Architecture '92, Reflection and Metalevel Architectures. RISE (Japan), ACM Sigplan, JSSST, IPSJ, November 1992.

[6] VisualWorks v1.0. ParcPlace Systems, 1992.

[7] ObjectWorks\Smalltalk v4.1. ParcPlace Systems, 1992.

[8] J.-P. Briot and P. Cointe. Programming with Explicit Metaclasses in Smalltalk-80. Proc. of OOPSLA'89, ACM Sigplan Notices, 24(10):419-432, October 1989.

[9] J. Malenfant, G. Lapalme and J. Vaucher. ObjVProlog: Metaclasses in Logic. Proc. of ECOOP'89, Cambridge U. Press, pp. 257-269.