Birds-Of-a-Feather: Reflection Meta-Objects, Classes, Metaclasses: Similitudes and Differences

Jean-Pierre Briot, Nicolas Graube*
Equipe Mixte L.I.T.P. & Rank Xerox France

Jacques Ferber
L.A.F.O.R.I.A.

Université Paris VI
Tour 45-55 Porte 209
4, Place Jussieu
75252 Paris cedex 05
France

Abstract

Reflection and Metaclasses are two very popular concepts in Object Oriented Languages based on Class taxonomy. However, if the second term is clearly described and documented, the first one is still surrounded by some artistic fog. This position paper is intended to bring into focus some new definitions in order to clarify the debate and lay a safe basis for further investigation. After proposing some definitions, we discuss several aspects of reflection. Then we briefly review the basic interests of reflection. Finally we will exhibit how reflection is already partially at work in class taxonomy-based languages. This gives us opportunity to compare the concepts of meta-object and metaclass.

1 Introduction.

The concept of reflection was engineered by Brian Smith in order to describe and control Lisp semantics in Lisp. It soon became very popular and rapidly spread outside the boundaries of the pure Lisp world.

Within Object Oriented Languages the concept of reflection was not clearly used nor described until Patti Maes' work on knowledge representation systems (3-KRS) and Watanabe's description of a reflective concurrent actor-based language (ABCL-R).

*This research was partly funded by the *GRECO de programmation du CNRS*.

Goals of reflection are description and control of implementation and activation of objects. However, the notion of meta-object still is not that clear for most popular paradigms, such as class taxonomy-based languages.

2 Meta-Object.

2.1 Definition.

Definition: 2.1 (Meta-Object.) *The meta-object is an explicit, consistent and usable representation of another object in terms of some object(s).*

The representation includes:

structure: data and physical implementation,

reactive abilities: message acceptance, processing and sending,

meaning: semantics, goal and finality.

A meta-object can fully or partially describe an object. Thus, an object could hold several meta-objects, each one depicting a specific point of view. In the case of a complete meta-object, all information held by the object is contained by the meta-object with no loss.

2.2 Consistency.

Information described by a meta-object must always be consistent with the object. This implies a bidirectional and causal connection between them.

2.3 Level and Meta-Level distinction.

An object holds two kinds of information, explicit and implicit. The object is conscious about its explicit information (e.g., a point knows the value of its "`x`" coordinate) while it unconsciously undergoes implementation decisions (e.g., a point implementation holds at its second location the value of its "`x`" coordinate).

Following the definition, implicit information is manipulable within the meta-object, while only explicit information is at the object level.

A good programmation methodology is to clearly separate the object level and the meta-object level.

One can note that Smalltalk-80 does not clearly draw these boundaries as it is possible for any object to use both level and meta-level tools (e.g., a point can access its "`x`" coordinate both through object level access (message passing or implicit reference) and by using the method `instVarAt:` to access the corresponding value which requires the knowledge of the related index).

3 Goals and Uses of reflection.

4 Reflection at work in class taxonomy-based languages.

The notion of class was introduced for abstraction and factorization of objects which do similar things.

4.1 Classes.

A class describes both structural components and the behavior of its instances. The structural components depicted by the class are usually known as instance variables or slots. These elements are only partially related with the physical implementation of the instances. In terms of behavior, a class usually describes the reaction of instances regarding certain messages sent to its instances or regarding the application of certain generic functions. On the other hand a class does not depict the way messages or generic functions are applied to its instances, nor the way method lookup is achieved.

4.2 Metaclasses.

A metaclass is a class whose instances are classes. The goal of metaclasses is to bring classes up to first class object level. Metaclasses are themselves classes. ObjVLisp and CLOS are such systems.

The main advantages of such a consideration are:

uniformity: classes and instances are treated in the same way,

parametrization: possibility of adding both new instance variables and methods to classes, in order to extend their functionnalities,

implementation control: metaclasses control the allocation of instances of their instances, thus the physical description of these objects.

4.3 Metaclasses, Classes, Instances and Reflection.

Once the duties of class and metaclass regarding instances have been clarified, a major step is to describe the relations that may be drawn between these three entities and the meta-object.

Reflection has to be split into two distinct fields: structural reflection and behavioral reflection. Following the definition, structural reflection is only concerned with the ability of depicting structural organisation of objects, whereas behavioral reflection is focused upon the description of all the "activity" part such as message passing handling.

4.3.1 Structural Reflection.

This kind of reflection is the easiest one to highlight within such systems.

The structure of an object is described by its class. The implementation of an object is described by its metaclass. Thus together these two components can be considered as a **implicit** and partial structural meta-object. Note that this meta-object is shared between all instances of the class, reflecting the factorization brought by classes in class taxonomy-based systems. A major consequence of this remark is that the modification of this meta-object (e.g., adding instance variable, changing structural representation) must be propagated among all instances of the class.

This consequence is already present within both Smalltalk-80 and CLOS as the change of a class triggers the updating of its instances. CLOS exposes this mechanism through the `change-class` generic function while Smalltalk-80 keeps it hidden. Within these systems, an instance cannot directly modify its class nor its metaclass, thus the causal connection between the object and its meta-object is only to be set from the meta-object to the object(s).

5 Behavioral Reflection.

Aside from the fact that class describes the behavior of instances regarding message passing or generic functions, in standard systems very little control of message application is provided.

In Smalltalk-80, for efficiency reasons both message acceptance, sending and method lookup mechanism are frozen within the virtual machine. In order to gain control on these mechanisms the implementor needs to use tricks, usually based on the use of the `doesNotUnderstand:` method (see for instance Foote's OOPSLA paper). It is also possible to provide a more general and complete behavioral reflection by using already existing tools such as "`thisContext`"

In CLOS the situation is a little bit more sheerful as several "hooks" are provided in order to enable modification of some parts of the generic function execution protocol.

6 Conclusion.

This short study was intended to bring into focus the relationship between the concepts of meta-object and class/metaclass. We showed that the class/metaclass as a whole is equivalent to a partial structural meta-object, shared by instances of the class.