REFLECTION

Nada Amin (na482)
Jeremy Yallop (jdy22)
Reflection is an entity’s integral ability to represent, operate on, and otherwise deal with itself in the same way it represents, operates on, and deals with its primary subject matter.

-Brian Cantwell Smith
Introspection is the ability of a program to simply reason about reifications of otherwise implicit aspects of itself or of the programming language implementation.
Intercession is the ability of a program to actually act upon reification of otherwise implicit aspects of itself or of the programming language implementation.
Structural reflection is the ability of a program to access a representation of its structure, as it is defined in the programming language.
Behavioral reflection is the ability of a program to access a dynamic representation of itself, that is to say, of the operational execution of the program as it is defined by the programming language implementation.
REFLECTIVE TOWERS

Of Interpreters
Reification is the process by which the state of the interpreter is passed to the program itself, suitably packaged (reified) so that the program can manipulate it.
Reflection is the process by which program values are re-installed as the state of the interpreter.
The Art of the Metaobject Protocol

Gregor Kiczales
Jim des Rivieres
Daniel G. Bobrow
Metaobject protocols are interfaces to the language that gives users the ability to incrementally modify the language behaviour and implementation, as well as the ability to write programs within the language.
First the basic elements of the programming language — classes, methods, generic functions — are made accessible as objects. Because these objects represent fragments of a program, they are given the special name of metaobjects.
Second, individual decisions about the behavior of the language are encoded in a protocol operating these meta objects — a metaobject protocol.
Third, for each kind of metaobjects, a default class is created, which lays down the behaviour of the default language in the form of methods in the protocol.
(defclass position ()
  (x y))

many instances,
both slots always used

array-like representation

(defclass person ()
  (name age address ...))

many instances,
only a few slots used in any one instance

hash-table like representation
standard-class

instance structure

inheritance rules

make a virtual copy of the implementation

and

replace one (or more) components

hash-table-class

accessors

inheritance rules
IMPLEMENTATION

➤ (defclass hash-table-class (standard-class) ()

➤ (defmethod allocate-instance ((c hash-table-class))
   ...allocate a small hash table to store the slots...)

➤ (defmethod get-value ((c hash-table-class) instance slot-name)
   ...get the slot value out of the hash table...)

➤ (defmethod set-value ((c hash-table-class) instance slot-name new-value)
   ...store the slot value in the hash table...
MIRRORS (BRACHA & UNGAR)

➤ **Encapsulation**: meta-level facilities must encapsulate their implementation.

➤ **Stratification**: meta-level facilities must be separated from base-level functionality.

➤ **Ontological correspondence**: the ontology of meta-level facilities should correspond to the ontology of the language they manipulate.
REFLECTION DANGERS

Examples in Java
CONCLUSIONS

➤ Reflection is the ability to reason about and act upon a representation of a program, be it dynamic and static.

➤ For MOPs, reflection enable a design space of options to be presented to the user, rather than a singular option.

➤ In general, reflection provides either too much or too little.

➤ Too much, because it enables changes that break the integrity of the system.

➤ Too little, because it doesn’t always accommodate changes not anticipated by the designers of the reflective system.