Mop

Greg

03 July 2013

Contents

1 Mop **1**
- 1.1 MOP Briefing . . . 1
  - 1.1.1 8 introspective procedures . . . 3
  - 1.1.2 9 intercessory generics . . . 3
- 1.2 Tiny-Clos Scheme . . . 3
- 1.3 Tiny-Clos Common Lisp . . . 3
- 1.4 Closette . . . 4

1 Mop

Notes on the implementation of three CLOS MOPs.

The practical end goal is to understand the CLOS MOP it so well that I will be able to port it to something like the JSCL Lisp-to-Javascript compiler.

https://github.com/davazp/jscl http://davazp.net/jscl/jscl.html

1.1 MOP Briefing

The **fundamental summary** for the purposes of this document is the description in Kiczales-1994.

> [Tiny Clos is] (a) very simple CLOS-like language, embedded in Scheme, with a simple MOP. The features of the default base language are:
>
> ; Classes, with instance slots, but no slot options. ; Multiple-inheritance. ; Generic functions with multi-methods and class

specializers only. ; Primary methods and call-next-method; no other method combination. ; Uses Scheme's lexical scoping facilities as the class and generic function naming mechanism. Another way of saying this is that class, generic function and methods are first-class (meta)objects.

While the MOP is simple, it is essentially equal in power to both MOPs in AMOP. This implementation is not at all optimized, but the MOP is designed so that it can be optimized.

This MOP actually allows better optimization of slot access extensions than those in AMOP.

In addition to calling a generic, the entry points to the default base language are:

(MAKE-CLASS list-of-superclasses list-of-slot-names) (MAKE-GENERIC) (MAKE-METHOD list-of-specializers procedure) (ADD-METHOD generic method)

(MAKE class .initargs) (INITIALIZE instance initargs) ;Add methods to this, ;don't call it directly. (SLOT-REF object slot-name) (SLOT-SET! object slot-name new-value)

So, for example, one might do: (define <position> (make-class (list <object>) (list 'x 'y))) (add-method initialize (make-method (list <position>) (lambda (call-next-method pos initargs) (for-each (lambda (initarg-name slot-name) (slot-set! pos slot-name (get1 initargs initarg-name 0))) '(x y) '(x y))))) (set! p1 (make <position> 'x 1 'y 3))

NOTE! Do not use EQUAL? to compare objects! Use EQ? or some hand written procedure. Objects have a pointer to their class, and classes are circular structures, and ...

The intospective part of the MOP looks like the following. Note that these are ordinary procedures, not generics.

CLASS-OF

CLASS-DIRECT-SUPERS CLASS-DIRECT-SLOTS CLASS-CPL CLASS-SLOTS

GENERIC-METHODS

METHOD-SPECIALIZER METHOD-PROCEDURE

The intercessory protocol looks like (generics in uppercase):

make ALLOCATE-INSTANCE INITIALIZE (really a base-level generic)

class initialization COMPUTE-CPL COMPUTE-SLOTS COMPUTE-GETTER-AND-SETTER

add-method COMPUTE-APPLY-GENERIC COMPUTE-METHODS COMPUTE-METHOD-MORE-SPECIFIC? COMPUTE-APPLY-METHODS

So that's it. We will be analyzing this quote throughout.

Now from the fundamental summary we can extract the core.

A quote from "tiny-announce.text" associated with 1994-Kiczales states:

"The MOP in Tiny CLOS is very simple – 8 introspective procedures and 9 intercessory generics..."

With this information, we can start getting a good overall view of the MOP.

The MOP has 17 low-level procedures/generics. That's it.

These 17 low-level objects are divided into two categories:

- Introspective procedures
- Intercessory generics

1.1.1 8 introspective procedures

CLASS-OF CLASS-DIRECT-SUPERS CLASS-DIRECT-SLOTS CLASS-CPL CLASS-SLOTS GENERIC-METHODS METHOD-SPECIALIZERS METHOD-PROCEDURE

1.1.2 9 intercessory generics

make ALLOCATE-INSTANCE INITIALIZE

class initialization COMPUTE-CPL COMPUTE-SLOTS COMPUTE-GETTER-AND-SETTER

add-method COMPUTE-APPLY-GENERIC COMPUTE-METHODS COMPUTE-METHOD-MORE-SPECIFIC? COMPUTE-APPLY-METHODS

1.2 Tiny-Clos Scheme

The first version of Tiny CLOS we will look at is the Scheme version written by Gregor Kiczales. The latest update I have on this is 8/9/94. It was found at the following weblink:

ftp://ftp.parc.xerox.com/pub/mops/tiny/tiny-clos.scm

1.3 Tiny-Clos Common Lisp

The second version of Tiny CLOS we will look at is the Common Lisp version written by Gregor Kiczales.

1.4 Closette

Closette is the full blooded Common Lisp version that is described in Gregor Kiczales, Jim des Rivieres and Daniel G. Bobrow's classic work *The Art of the Metaobject Protocol*, or *AMOP* as it is known throughout the Lisp community.