Generalizers

New Metaobjects for Generalized Dispatch

Christophe Rhodes, Jan Moringen, David Lichteblau

5th May 2014

Goldsmiths
UNIVERSITY OF LONDON

output

Method dispatch

CL algorithm

7.6.6.1 Determining the Effective Method

1. Selecting the Applicable Methods
2. Sorting the Applicable Methods by Precedence Order
3. Applying method combination to the sorted list of applicable methods

Method dispatch

MOP for standard-generic-function

- `compute-discriminating-function`
- `compute-applicable-methods`
- `compute-effective-method`

Method dispatch

MOP for standard-generic-function

- `compute-discriminating-function`

- `compute-applicable-methods`
- `compute-effective-method`
- invoke the effective method somehow

Method dispatch

MOP for standard-generic-function

- `compute-discriminating-function`
- `compute-applicable-methods-using-classes`
- `compute-applicable-methods`
- `compute-effective-method`
- invoke the effective method somehow

Method dispatch

compute-applicable-methods-using-classes

`compute-applicable-methods-using-classes` (gf list)

- *gf* argument is the generic function being called
- *list* argument is a list of the classes of the objects in the required-argument position

Method dispatch

compute-applicable-methods-using-classes

`compute-applicable-methods-using-classes` (gf list)

- *gf* argument is the generic function being called
- *list* argument is a list of the classes of the objects in the required-argument position

Computes sorted list of applicable methods of the generic function

- ... or defers to `compute-applicable-methods`

Method dispatch

compute-applicable-methods-using-classes

`compute-applicable-methods-using-classes` (gf list)

- *gf* argument is the generic function being called
- *list* argument is a list of the classes of the objects in the required-argument position

Computes sorted list of applicable methods of the generic function

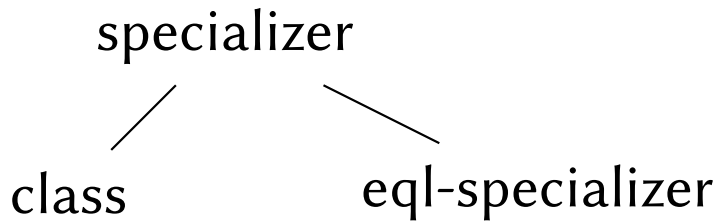
- ... or defers to `compute-applicable-methods`

If `c-a-m-u-c` succeeds, its return value is usable for all actual arguments to the generic function of the same classes.

- effective method can be cached and reused!

Method dispatch

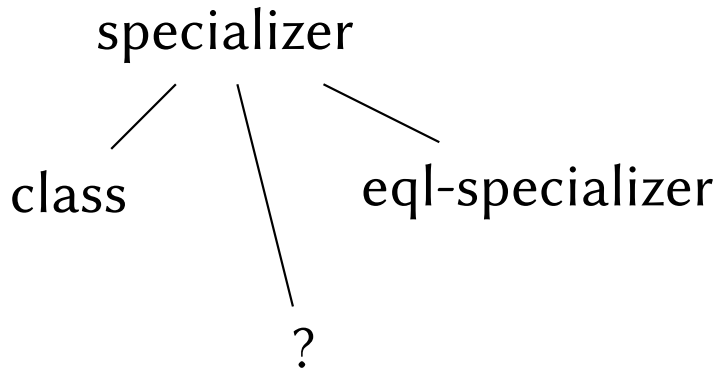
MOP class hierarchy



Jim Newton and Christophe Rhodes, *Custom Specializers in Object-Oriented Lisp*, 2008

Method dispatch

MOP class hierarchy



Jim Newton and Christophe Rhodes, *Custom Specializers in Object-Oriented Lisp*, 2008

Custom specializers

Example: dispatch on signum

```
(defgeneric fact (n)
  (:generic-function-class signum-generic-function))

(defmethod fact ((n (signum 1)))
  (* n (fact (1- n))))
(defmethod fact ((n (signum 0)))
  1)

(fact 0) ; => 1
(fact 10) ; => 3628800
(fact -1) ; error "no applicable method"
```

Generalizers

How to replace `compute-applicable-methods-using-classes` for custom specializers?

1st try: `compute-applicable-methods-using-specializers`

- does not work!
- (does not even make sense)

Generalizers

How to replace `compute-applicable-methods-using-classes` for custom specializers?

1st try: `compute-applicable-methods-using-specializers`

- does not work!
- (does not even make sense)

2nd try: distinguish between class as specializer (restrictive) and class as equivalence class (expansive)

- works!
- motivates the `generalizer` metaobject

Generalizers

The Generalizer protocol

- `generalizer` [class]
- `generalizer-of-using-class (gf ob)` [gf]

Generalizers

The Generalizer protocol

- `generalizer` [class]
- `generalizer-of-using-class (gf ob)` [gf]
- `specializer-accepts-generalizer-p (gf sp ge)` [gf]
- `specializer-accepts-p (sp ob)` [gf]

Generalizers

The Generalizer protocol

- `generalizer` [class]
- `generalizer-of-using-class (gf ob)` [gf]
- `specializer-accepts-generalizer-p (gf sp ge)` [gf]
- `specializer-accepts-p (sp ob)` [gf]
- `specializer< (gf sp1 sp2 ge)` [gf]

Generalizers

The Generalizer protocol

- `generalizer` [class]
- `generalizer-of-using-class (gf ob)` [gf]
- `specializer-accepts-generalizer-p (gf sp ge)` [gf]
- `specializer-accepts-p (sp ob)` [gf]
- `specializer< (gf sp1 sp2 ge)` [gf]
- `generalizer-equal-hash-key (gf ge)` [gf]

Generalizer protocol

Example: dispatch on signum revisited

```
(defclass signum-generalizer (generalizer)
  ((%signum :reader %signum :initarg :signum)))

(defmethod generalizer-of-using-class
    ((gf signum-generic-function) (arg real))
  (make-instance 'signum-generalizer :signum (signum arg)))
(defmethod generalizer-equal-hash-key
    ((gf signum-generic-function) (g signum-generalizer))
  (%signum g))

(defmethod specializer-accepts-generalizer-p
    ((gf signum-generic-function)
     (s signum-specializer) (g signum-generalizer))
  (if (= (%signum s) (%signum g))
      (values t t)
      (values nil t)))
(defmethod specializer-accepts-p ((s signum-specializer) o)
  (and (realp o) (= (%signum s) (signum o))))
```

Generalizer protocol

Example: HTTP content negotiation

```
(defgeneric foo (request)
  (:generic-function-class accept-generic-function))
(defmethod foo ((request t)) (http:406 request))

(defmethod foo ((request (accept "text/html")))
  "<!DOCTYPE html>
<html><head><title>Foo</title></head>
<body><p>Foo</p></body></html>")

(defmethod foo ((request (accept "text/turtle")))
  "@prefix foo: <http://example.org/ns#> .
@prefix : <http://other.example.org/ns#> .
foo:bar foo: : .")

(foo "text/html,application/xml;q=0.9,*/*;q=0.8")
  ; => text/html version
(foo "text/turtle") ; => text/turtle version
```

Generalizer protocol

Example: HTTP content negotiation

- non-trivial non-standard dispatch
- distinct specializer and generalizer objects
- dispatch decoupled from web server implementation:
  - one new method on `specializer-accepts-p`
  - one new method on `generalizer-of-using-class`

Generalizer protocol

Efficiency

Signum Specializers:

implementation	time (µs/call)	overhead
function	0.6	
standard-gf/fixnum	1.2	+100%
signum-gf/one-arg-cache	7.5	+1100%
signum-gf	23	+3800%
signum-gf/no-cache	240	+41000%

Related Work

- predicate dispatch
- filtered functions
- layered functions
- prototype dispatch

Michael Ernst, Craig Kaplan, and Craig Chambers. *Predicate dispatching: A unified theory of dispatch*, 1998.

Related Work

- predicate dispatch
- filtered functions
- layered functions
- prototype dispatch

Pascal Costanza, Charlotte Herzeel, Jorge Vallejos, and Theo D'Hondt. *Filtered Dispatch*, 2008.

Related Work

- predicate dispatch
- filtered functions
- layered functions
- prototype dispatch

Robert Hirschfeld, Pascal Costanza, and Oscar Nierstrasz. *Context-oriented programming*, 2008

Related Work

- predicate dispatch
- filtered functions
- layered functions
- prototype dispatch

Lee Salzman and Jonathan Aldrich. *Prototypes with Multiple Dispatch: An Expressive and Dynamic Object Model*, 2005.

Conclusions

Customizing specializers is now:

- easier (thanks to a simple protocol with local computations);
- better-performing (10-30 times faster than naïve implementation, though still 2–6 times slower than standard dispatch);
- straightforwardly available (simply load into a running SBCL).

Conclusions

Customizing specializers is now:

- easier (thanks to a simple protocol with local computations);
- better-performing (10-30 times faster than naïve implementation, though still 2–6 times slower than standard dispatch);
- straightforwardly available (simply load into a running SBCL).

Currently working on:

- pattern specializers (optima) with automatic variable bindings;
- more flexibility on cacheing / dispatch computation.