Method Combinators

ELS 2018

Didier Verna
EPITA / LRDE

didier@lrde.epita.fr
Introduction

- CLOS improvements over mainstream object systems
  - Multiple dispatch
    - *Increased SOC: polymorphism / inheritance*
  - MOP
    - *Homogeneous behavioral reflection*
  - Method combinations
    - *Increased SOC: methods / dispatch*

- Standardization drawbacks
  - Method combinations underspecified
    - *Considered not mature enough*
  - MOP only a later addition
    - *Unclear or contradictory protocols*
Introduction

Issues

SBCL

Combinators

CGFs

Alt. MCs

Perfs

Conclusion

Plan

Method Combinations Issues

The Case of SBCL

Method Combinators

Combined Generic Functions

Alternative Combinators

Performance
Method Combinations Issues

The Case of SBCL

Method Combinators

Combined Generic Functions

Alternative Combinators

Performance
Orthogonality

Short combination example

```lisp
(defun details (human)
  (:method-combination append :most-specific-last)
  (:method append ((human human)) ...)
  (:method append ((employee employee)) ...))
```

- **Problems**
  - Method qualification required
    
    *Combination change impractical*
  - Except for the option
    
    *Inconsistent*
  - No :before or :after methods
    
    *No good reason*

- **Workaround: long method combinations**
Portable specialization impossible

At least one implementation-specific (sub)class

Unclear nature (classes vs. instances)

Mix of define / call-time parametrization
Lookup (MOP)

```
find-method-combination gf name options
```

- “called to determine the combination object used by a generic function”
  - What are `name` and `options` for?
  - Error behavior?
  - There already is `generic-function-method-combination`
Generic function invocation protocol (MOP)

compute-effective-method gf combination methods

- What is combination for?
- Caching policy unspecified

Contrary to applicable methods
Plan

Method Combinations Issues

The Case of SBCL

Method Combinators

Combined Generic Functions

Alternative Combinators

Performance
Method combination classes hierarchy

- **standard-method-combination**
  - type-name
  - options
- **short-method-combination**
  - operator
  - identity-with-one-argument
- **long-method-combination**
  - function
  - args-lambda-list

- options: use-time (:method-combination options)
- Below: define-time
Creation

(\texttt{define-method-combination name option*})

\texttt{(find-method-combination \texttt{gf} (eql name) options)}

\textbf{No global namespace}

- One method combination object per generic function
- Redefinitions don’t affect existing generic functions

\texttt{find-method-combination \neq \text{the expected or the specified}}
Long method combination functions

*long-method-combination-functions*

<table>
<thead>
<tr>
<th>long-method-combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>function</td>
</tr>
<tr>
<td>args-lambda-list</td>
</tr>
</tbody>
</table>

- Similar behavior, one additional oddity
  - Local method combination objects
  - Global method combination functions
Code

```
(define-method-combination my-progn ()
  ((primary () :order :most-specific-first :required t))
  `(progn ,@(mapcar (lambda (method)
      `(call-method ,method))
    primary)))

(defun test (i) (:method-combination my-progn)
  (:method ((i number)) (print 'number))
  (:method ((i fixnum)) (print 'fixnum)))
```

REPL

```
CL-USER> (test 1)
FIXNUM
NUMBER
```
Code

(define-method-combination my-progn ()
  ((primary () :order :most-specific-last :required t))
  `(progn ,@(mapcar (lambda (method)
                    `(call-method ,method))
               primary)))

REPL

CL-USER> (test 1)
FIXNUM
NUMBER
Code

(defmethod test ((i float)) (print 'float))

REPL

CL-USER> (test 1.5)           CL-USER> (test 1)
NUMBER FIXNUM
FLOAT NUMBER
Method Combinations Issues

The Case of SBCL

Method Combinators

Combined Generic Functions

Alternative Combinators

Performance
Introduction

Issues

SBCL

Combinators

CGFs

Alt. MCs

Perfs

Conclusion

Overview

Classes

- short-method-combination
- long-method-combination
- method-combinator-mixin
- clients
- short-method-combinator
- long-method-combinator

- Stored in a global hash table
- [setf] find-method-combinator

Method Combinators / ELS 2018 – Didier Verna
Protocols

In 3 layers

1. define-[short|long]-method-combinator
2. ensure-[short|long]-method-combinator
3. ensure-[short|long]-method-combinator-using-class
In 4 steps

- define a regular combination
  (find-method-combination)
- retrieve it
  (change-class)
- make it combinator
  (setf find-method-combinator)
- store it

Note: regular combination injection
Method Combinations Issues

The Case of SBCL

Method Combinators

Combined Generic Functions

Alternative Combinators

Performance
Overview

Classes

- `standard-generic-function`
- `funcallable-standard-class`
- `combined-generic-function`
- `functions`

Wrappers

```lisp
(defcombined cgf (args...)
  (:method-combinator mc)
  ...)
```
Method Combinator Management

- **Initialization**
  
  \[
  \text{(defmethod \text{find-method-combination} \ (\text{cfg-class-prototype} \ldots))} \\
  \text{(find-method-combinator \ldots))}
  \]

- **Sanitation**
  
  \[
  \text{(defmethod \text{find-method-combination} \ (\text{cfg} \ldots)} \\
  \text{(method-combinator \text{cfg})} \text{#|or mismatch error|#)}
  \]

- **Updating**
  
  \[
  \text{(change-method-combinator \text{cfg} \text{method-combinator})}
  \]
Client Maintenance

- Client registration:
  \((\text{[re]initialize-instance cgf ...})\)

- Client updating:
  \((\text{reinitialize-instance mc ...})\)
  \((u-i-f-d-c mc ...\))

New protocol

- \(\text{make-clients-obsolete}\)

- \(\text{update-combined-generic-function-for-redefined-method-combinator}\)
Method Combinations Issues

The Case of SBCL

Method Combinators

Combined Generic Functions

Alternative Combinators

Performance
Overview

- **Idea:** generic functions / combinators complete decoupling
- **Use:** ≠ logical method combinations, selected methods etc.
- **Note:** already possible, but extremely costly
  - 2 calls to reinitialize-instance

Protocols

```
(call-with-combinator (find-method-combinator 'combinator)
  #'func arg1 arg2 ...)
```

```
(call/cb combinator func arg1 arg2 ...)
```

```
#!combinator(func arg1 arg2 ...)
```
Optimization

- Discriminating functions caches
- Client maintenance aware of them
- Cost
  - First alternative call: as before
  - Next: 1 or 2 hashtable lookups

**Warning:** discriminating functions must close over all caches!
Method Combinations Issues

The Case of SBCL

Method Combinators

Combined Generic Functions

Alternative Combinators

Performance
Performance

Method Combinators / ELS 2018 – Didier Verna
Conclusion

- Method combinations are powerful yet underspecified
- Method combinators improve their consistency
- Code available on GitHub

Perspectives

- Refine / properly package implementation
- Port to other compilers
- Experiment with “floating” floating methods