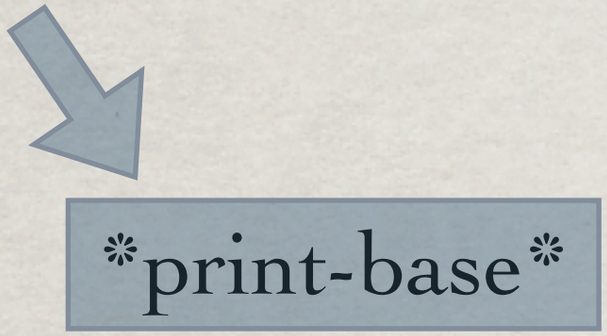
How to Make Lisp More Special

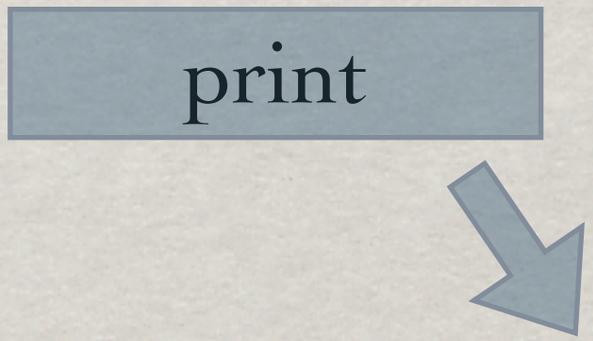
Pascal Costanza
VU Brussel

Dynamic Scoping

- Common Lisp:
  - “special variables”
    - Example: *print-base*
- Scheme:
  - with-output-to-file
  - “fluid variables”, parameter objects, etc.

Dynamic Scoping





Dynamic Scoping - Definitions in CLtL2

- Lexical scope: References may occur only within portions textually contained within the establishing construct.

- Indefinite scope:
  References may occur anywhere.

Dynamic Scoping - Definitions in CLtL2

- Dynamic extent: References may occur in the interval between establishment and disestablishment of an entity, obeying a stack-like discipline.

- Indefinite extent: The entity exists as long as the possibility of reference remains.

Dynamic Scoping - Definitions in CLtL2

- "Dynamic scope" is strictly a misnomer.
- Nevertheless, it is useful and traditionally means "indefinite scope & dynamic extent."

Dynamic Scoping as the Essence of AOP

- "A Simple Telecom Example"
  (from the AspectJ Programming Guide at http://eclipse.org/aspectj/)
  - Classes Customer - Call - LongDistance & Local Connection
  - Aspects Timing & Billing

Dynamic Scoping as the Essence of AOP

Compiling and Running

The files timing.lst and billing.lst contain file lists for the timing and billing configurations. To build and run the application with only the timing feature, go to the directory examples and type:

```
ajc -argfile telecom/timing.lst
java telecom.TimingSimulation
```

To build and run the application with the timing and billing features, go to the directory examples and type:

```
ajc -argfile telecom/billing.lst
java telecom.BillingSimulation
```

Dynamic Scoping as the Essence of AOP

- (with-active-aspects (timing)
    (timing-simulation))

- (with-active-aspects (timing billing)
    (billing-simulation))

- ...but with intermediate compilation...

Present & Future

- AspectL: AOP for Common Lisp
- Closer to MOP: Compatibility layer for Allegro, CLISP, CMUCL, LispWorks, MCL, OpenMCL, SBCL, and counting...
- ContextL: Context-Oriented Programming

Now

- The DLETF Framework
- An example: Special classes
- How is this implemented?

The DLETF Framework

- Recall SETF in Common Lisp:
  (setf (person-name p) “Pascal”)

- We want the same for bindings:
  (letf (((person-name p) “Pascal”)) ...)

- Let’s make it explicitly dynamically scoped:
  (dletf (((person-name p) “Pascal”)) ...)

The DLETF Framework

- An example:
  (similar to what can be done in CLIM)

```
(dletf (((medium-ink medium)    +red+)
        ((medium-style medium) +bold+)))
  (draw-line medium x1 y1 x2 y2))
```

The DLETF Framework

- DLETF itself is "only" a framework.
- Special classes are implemented by a metaclass that uses the hooks of DLETF.
- Other "plugins" are also possible.
  (lists, arrays, structures, hashtables, ...)

Implementation

- LETF on Lisp Machines
- LETF on "stock hardware":
  - Global side effects + unwind-protect
    (That's not what we want!)
- LETF vs. LETF-GLOBALLY

LETF-GLOBALLY

```
(let ((temp1 (medium-ink m))
      (temp2 (medium-style m)))
  (unwind-protect
      (progn (setf (medium-ink m) +red+
                   (medium-style m) +bold+)
             ...)
    (setf (medium-ink m) temp1
          (medium-style m) temp2)))
```

Implementation with progv

- From the HyperSpec:
  - "progv allows binding one or more dynamic variables whose names may be determined at runtime."
  - "The bindings of the dynamic variables are undone on exit from progv."
  - "[...] it provides a handle on the mechanism for dynamic variables."

The DLETF Protocol

- Store "special" symbols instead of values.
- Bind values as symbol values.
- Access the values if *symbol-access* is nil.
- Access the symbols otherwise.

The DLETF Protocol

```
(dletf (((medium-ink m) +red+)
        ((medium-style m) +bold+))      ...)
```

...expands to...

```
(progv (let ((*symbol-access* t))
         (list (medium-ink m)
               (medium-style m)))
       (list +red+ +bold+))                 ...)
```

The SPECIAL-CLASS Metaclass

```
(defclass medium ()
  ((ink :accessor medium-ink :special t)
   (style :accesser medium-style :special t))
  (:metaclass special-class))
```

The SPECIAL-CLASS Metaclass

```
(defmethod slot-value-using-class
  ((class special-class)
   object
   (slot special-effective-slot-definition))
  (let ((slot-symbol (call-next-method)))
    (cond (*symbol-access* slot-symbol)
          ((boundp slot-symbol)
           (symbol-value slot-symbol))
          (t (slot-unbound ...)))))
```

Some Technical Issues

- Slot initialization may bypass the slot accessors. Fixed via shared-initialize.
- Slots can be changed from non-special to special, but not vice versa. (Conversion from one binding to multiple bindings is easy, the other way around is not!)

Other Data Structures

- Arrays, lists, structures, etc., do not provide metaobject protocols.
- Instead: Shadow symbols of the common-lisp package. (see paper)

Efficiency Concerns

- Dynamic Scoping:
  shallow binding vs. deep binding vs.
  acquaintance vectors
- Double indirection: may not hurt.
- Slot access: only special slots are affected.

Summary

- DLETF part of AspectL:
  http://common-lisp.net/project/aspectl
  - also special-function, based on DLETF
- DLETF will also be part of ContextL:
  http://common-lisp.net/project/closer
- More to come...

The End