Lightweight Languages as Lightweight Operating Systems

Matthew Flatt

PLT
University of Utah
Premise

programming language = operating system
Premise

programming language   =

operating system

formal semantics for either is a virtual machine
Operating Systems vs Programming Languages

OS

virtualize machine for non-interference

PL

virtualize machine for expressiveness
Operating Systems vs Programming Languages

**OS**

- virtualize machine for non-interference

  ⇒ isolation

**PL**

- virtualize machine for expressiveness

  ⇒ cooperation
Operating Systems vs Programming Languages

**OS**
- virtualize machine for non-interference

$\Rightarrow$ **isolation**

**PL**
- virtualize machine for expressiveness

$\Rightarrow$ **cooperation**
Operating Systems vs Programming Languages

- **OS**: virtualize machine for non-interference
  - $\Rightarrow$ isolation

- **PL**: virtualize machine for expressiveness
  - $\Rightarrow$ cooperation
Programming with Processes

PLT Installer

- updating bin/icfp-robot-client
- updating bin/simple-icfp-robot
- updating bin/setup-plt
- updating bin/framework-test-engine
- updating bin/framework-test
- updating bin/web-server
- updating bin/web-server-text
- updating bin/pdf-slatex
- updating bin/slideshow

setup-plt: Setup version is 202.5
setup-plt: PLT home directory is /home/mflatt/proj/plt
setup-plt: Collecting Paths is /home/mflatt/proj/plt/collect
Programming with Processes
Programming with Processes

From: gregs@ai.mit.edu (Gregory T. Sullivan)
To: LL1 Discussion <lll-discuss@ai.mit.edu>
Subject: LL2, Sat. Nov. 9, 2002 - webcast info.
Date: 06 Nov 2002 11:51:57 -0500

Thanks to the generous sponsorship of Microsoft and Dr. Dobbs Journal, we will be able to webcast the LL2 proceedings this coming Saturday, Nov. 9. The morning session (10am to 12:35pm EST) can be viewed at:

http://web.mit.edu/webcast/ailab/mit-ll2-s1-09nov02-80k.ram

and the afternoon, from 1:45pm to 6pm will be at:

http://web.mit.edu/webcast/ailab/mit-ll2-s2-09nov02-80k.ram

No new mail
Manifesto

What:

- PL designers should think more as OS designers
- Maintain PL bias toward cooperation

How:

- Break process abstraction into pieces
- Replace isolation with enforced abstraction
Outline

- Motivation and Manifesto
- PLT Scheme as an Example
  - Threads
  - Parameters
  - Eventspaces
  - Custodians
  - Inspectors
- Assembling the Pieces
Threads

Concurrent execution

```
(require "spin-display.scm")  eval

(define (spin)
  (rotate-a-little)
  (sleep 0.1)
  (spin)
  (spin))

(define spinner (thread spin))  eval

(kill-thread spinner)  eval
```
Parameters (a.k.a. Fluid Variables)

Thread-local state

(printf "Hello\n")
(fprintf (current-output-port) "Hola\n")
(fprintf (current-error-port) "Goodbye\n")
(error "Ciao")

(eval

(parameterize ((current-error-port (current-output-port)))
  (error "Au Revoir"))

(eval

(parameterize ((current-error-port (current-output-port)))
  (thread
    (lambda ()
      (error "Zai Jian"))))

(eval
Eventspaces

Concurrent GUIs

(thread (lambda () (message-box "One" "Hi")))
(thread (lambda () (message-box "Two" "Bye")))

(eval

(thread (lambda () (message-box "One" "Hi")))
(parameterize ((current-eventspace (make-eventspace)))
  (thread (lambda () (message-box "Two" "Bye"))))

(eval
Custodians

Termination and clean-up

(define c (make-custodian))
(parameterize ((current-custodian c)) ...
  ...
 eval)

(custodian-shutdown-all c) eval
Inspectors

Debugging access to data

```
(define-struct fish (color weight))
(define eddie (make-fish 'red 100))
(print eddie)
```

```
(define senior-inspector (current-inspector))
(parameterize ([current-inspector (make-inspector)])
  (define-struct fish (color weight))
  (define eddie (make-fish 'red 100))
  (parameterize ([current-inspector senior-inspector])
    (print eddie)))
```
Etc.

- Security Guards

  *Resource access control*

- Namespaces

  *Global bindings*

- Will Executors

  *Timing of finalizations*
Outline

- Motivation and Manifesto
- PLT Scheme as an Example
- Assembling the Pieces
  - SchemeEsq, a mini DrScheme [ICFP99]
(define frame
  (instantiate frame% ()
    [label "SchemeEsq"]
    [width 400] [height 175]))

(send frame show #t)

(eval)
GUI - Reset Button

(instantiate button% ()
    [label "Reset"]
    [parent frame]
    [callback (lambda (b e) (reset-program))])
GUI - Interaction Area

(define repl-display-canvas
  (instantiate editor-canvas% ()
    [parent frame]))

eval
(define esq-text%
    (class text% ... (evaluate str) ...))

(define repl-editor (instantiate esq-text% ()))
(send repl-display-canvas set-editor repl-editor)
(define (evaluate expr-str)
  (thread
    (lambda ()
      (print (eval (read (open-input-string expr-str))))
      (newline)
      (send repl-editor new-prompt)))
  )
(define user-output-port
  (make-custom-output-port ... repl-editor ...))

(define (evaluate expr-str)
  (parameterize (((current-output-port user-output-port))
                (thread
                 (lambda ()
                  (...)))))

eval
Evaluating GUls

(define user-eventspace (make-eventspace))

(define (evaluate expr-str)
  (parameterize ((current-output-port user-output-port)
    (current-eventspace user-eventspace))
    (thread
      (lambda ()
        ...)))))

(eval
Custodian for Evaluation

(define user-custodian (make-custodian))

(define user-eventspace
  (parameterize ([(current-custodian user-custodian)])
                (make-eventspace)))

(define (evaluate expr-str)
  (parameterize ([(current-output-port user-output-port)]
                 [(current-eventspace user-eventspace)]
                 [(current-custodian user-custodian)])
    (thread
     (lambda ()
       (eval ...)))))
(define (reset-program)
  (custodian-shutdown-all user-custodian)

  (parameterize ((current-custodian user-custodian))
    (set! user-eventspace (make-eventspace)))
  (send repl-editor reset))

(eval)
Inspecting Results

(define esq-inspector (current-inspector))
(define user-inspector (make-inspector))

(define (evaluate expr-str)
  (parameterize (...)
    (current-inspector user-inspector))

(thread
  (lambda ()
    (let ([v (eval ...)])
      (parameterize ((current-inspector esq-inspector))
        (print v))
      ...)))))
Conclusion

- Programmers need OS-like constructs in languages
  - run-time environment (e.g., script GUI wrapper, talk demos)
  - easy termination (e.g., server sessions, awry demos)
  - concurrency (e.g., browser cancel, mail client)
- Multiple language constructs for "process"
  - programmer can mix and match to balance isolation and cooperation