Knit, Chisel, Hack: Crafting with Guile Scheme

Andy Wingo ~ wingo@igalia.com
wingolog.org ~ @andywingo
I love craft!

Woodworking
Gardening
Grow-your-own
Brew-your-own
Knit-your-own
Sew-your-own
Roast-your-own
Repair-your-own
Build-your-own

Why?
Crafty pleasures

Making and building
Quality of result
Expressive aspect: creativity
Fitness to purpose
Increasing skill
what’s not crafty?
what’s the difference?

Craft is produced on human scale (hand tools)
Craft is made to fit (own clothes)
Craft touches roots (grow your own)
Craft is generative (wearables)
craft
/kraːft/
noun

1. an activity involving skill in making things by hand.
   "the craft of cobbling"
   synonyms: activity, pursuit, occupation, work, line, line of work, profession, job,
   business, line of business, trade, employment, position, post, situation,
   career, métier, vocation, calling, skill, field, province, walk of life; More

2. skill used in deceiving others.
   "her cousin was not her equal in guile and evasive craft"
   synonyms: cunning, craftiness, guile, wiliness, artfulness, deviousness, slyness,
   trickery, trickiness; More
craft

/kraːft/

noun

1. an activity involving skill in making things by hand.
   "the craft of cobbling"
   synonyms: activity, pursuit, occupation, work, line, line of work, profession, job,
   business, line of business, trade, employment, position, post, situation,
   career, métier, vocation, calling, skill, field, province, walk of life; More

2. skill used in deceiving others.
   "her cousin was not her equal in guile and evasive craft"
   synonyms: cunning, craftiness, guile, wiliness, artfulness, deviousness, slynness,
   trickery, trickiness; More
ohai!

Guile co-maintainer since 2009

Publicly fumbling towards good
Scheme compilers at wingolog.org

Thesis: Guile lets you build with craft
quick demo
scheme expressions

Constants: 1, "ohai"

Some constants need to be quoted:
'(peaches cream)

Functions: (lambda (a b) (+ a b))

Calls: (+ a b)

Sequences: (begin (foo) (bar))

If: (if (foo) (bar) (baz))

Lexicals: (let ((x (foo))) (+ x x))

That’s (pretty much) it!
repl
as
workbench

,profile
,disassemble
,break
,time
,expand
,optimize
.bt
,help
building and growing

How to take a small thing and make it bigger?

How to preserve the crafty quality as we add structure?
scripts

Do more by leveraging modules

(use-modules (ice-9 match)
  (web client))

(match (program-arguments)
  ((arg0 url)
    (call-with-values
      (lambda () (http-get url))
      (lambda (response body)
        (display body))))))
built-in modules

POSIX
Web (client, server, http bits)
I/O (Binary and textual, all encodings)
XML (and SXML)
Foreign function interface (C libraries and data)
Read the fine manual!
from scripts to programs

Script: Up to a few pages of code, uses modules to do its job
Program: It’s made of modules
System: No one knows what it does
from scripts to programs

Programs more rigid, to support more weight

Separate compilation for modular strength

Programs need tooling to manage change

- Keyword arguments for extensibility
- Warnings from compiler
- Facilities for deprecating and renaming interfaces
what's a scripting language anyway

A sloppy language with a slow implementation
A historical accident
guile’s speed bridges the gap

Allocation rate: 700-800 MB/s
Instruction retire rate: 400M-500M Inst/s
Startup time: 8.8ms
Minimum memory usage (64-bit): 2.15 MB
Sharing data via ELF
versus other langs

(All the caveats)

# Python 3
for i in range(0, 1000000000):
    pass

;;; Scheme
(let lp ((i 0))
  (when (< i #e1e9)
    (lp (1+ i))))

// C
for (long i = 0; i < 1000000000; i++)
    ;
versus other langs

Python 3: 81.2 cycles/iteration
Guile 2.0: 67.3 cycles/iteration
Guile 2.2: 12.1 cycles/iteration
gcc -O0: 5.66 cycles/iteration
gcc -O1: 0.812 cycles/iteration (3.7 IPC)
gcc -O2: friggin gcc
catching up on C

Native compilation coming in Guile 3
not catching up on C

Heap corruption
Stack smashing
Terrible errors
scale out

Guile has real threads and no GIL!
Processes too
But is it WEB SCALE?!?!?
tools for growth

Macros
Prompts
macros
extend
language
syntax

Different kinds of let: letpar, let-fresh,
Pattern matchers: match, sxml-match,
Constructors: SQL queries, nested structured records,
Instrumentation: assert-match, assert-index, logging
“Decorators”: define-deprecated, define-optimizer,
Cut a language to fit your problem
prompts

/home/wingo% ./prog

Two parts: system and user

Delimited by prompt
prompts

try {
    foo();
} catch (e) {
    bar();
}
prompts in guile scheme

Early exit
Coroutines
Nondeterminism
make a prompt

(use-modules (ice-9 control))

(% expr
  (lambda (k . args) #f))
make a prompt

(use-modules (ice-9 control))
(let ((tag (make-prompt-tag)))
  (call-with-prompt tag
    ;; Body:
    (lambda () expr)
    ;; Escape handler:
    (lambda (k . args) #f)))
prompts: (use-modules (ice-9 control))

(let (((tag (make-prompt-tag)))
   (call-with-prompt tag
     (lambda (tag)
       
       (lambda ()
         
         (+ 3
           
           (abort-to-prompt tag 42)))
         (lambda (k early-return-val)
           early-return-val)))

;; => 42
prompts: early exit

(define-module (my-module)
  #:use-module (ice-9 control)
  #:export (with-return))

(define-syntax-rule
  (with-return return body ...)
  (let ((t (make-prompt-tag)))
    (define (return . args)
      (apply abort-to-prompt t args))
    (call-with-prompt t
      (lambda () body ...)
      (lambda (k . rvals)
        (apply values rvals)))))
prompts:  
early exit

(use-modules (my-module))

(with-return return
  (+ 3 (return 42)))
;; => 42

(with-return return
  (map return '(1 2 3)))
;; => it depends :)

prompts: (use-modules (ice-9 control))
what (let ((tag (make-prompt-tag)))
about (call-with-prompt tag
    (lambda () ...)
    (lambda (k . args) ...))))
k? First argument to handler is continuation
Continuation is delimited by prompt
prompts:

what about k?

(use-modules (ice-9 control))

(define (f)
  (define tag (make-prompt-tag))
  (call-with-prompt tag
    (lambda ()
      (+ 3
        (abort-to-prompt tag)))
    (lambda (k) k)))

(let ((k (f)))
  (list (k 1) (k 2)))
;; => (4 5)
prompts: what about k?

When a delimited continuation suspends, the first argument to the handler is a function that can resume the continuation.

```
(let ((k (lambda (x) (+ 3 x))))
  (list (k 1) (k 2)))
;; => (4 5)
```

(For those of you that know call/cc: this kicks call/cc in the pants)
Prompts enable go-style concurrency.

Suspend “fibers” (like goroutines) when I/O would block.
Resume when I/O can proceed.
Ports to share data with world.
No need to adapt user code!

E.g. web server just works.

Channels to share objects with other fibers.
straight up network programs

(define (run-server)
  (match (accept socket)
    ((client . sockaddr)
      (spawn-fiber
        (lambda ()
          (serve-client client)))
      (run-server))))

(define (serve-client client)
  (match (read-line client)
    ((? eof-object?) #t)
     (line
      (put-string client line)
      (put-char client \newline)
      (serve-client client)))))
straight
up
network
programs

50K+ reqs/sec/core (ping)
10K+ reqs/sec/core (HTTP)
Handful of words per fiber
WEB SCALE!?!?!?!
Still lots of work to do
- work-stealing
- fairness
- nice debugging
- integration into Guile core
- external event loops

https://github.com/wingo/fibers
then deploy

Use Guix! https://gnu.org/s/guix/
Reproducible, deterministic, declarative clean builds, in Guile Scheme
Distribute Guile and all dependent libraries with your program
Run directly, or build VM, or (in future) docker container
godspeed!  https://gnu.org/s/guile/
#guile on freenode
Share what you make!
@andywingo