Hacking GNOME Applications in Scheme

'("Andy Wingo"
"23 February 2007"
"FOSDEM")
Why Scheme?

Not because it's common
Not because your boss tells you to
Not because your friends are doing it
Because it's fun!
Because it's not C

Garbage collection: no need for ref/unref, malloc/free

Runtime typing: no need for GTK_WIDGET (foo); no programmer-visible GValue
Because it is Scheme

Tail recursion is good for the mind

 Macros are fun

 Turns out the parentheses aren't so bad
Class libraries + Scheme
(gnome . scheme)

Goal of a language binding:

Minimize 'impedance mismatch' between two stacks.
Scheme basics

Prefix notation: verb first

3 + 8  ----  (+ 3 8)
getuid()  ----  (getuid)
Object-oriented Scheme?

Prefix notation: verb first!

```plaintext
bin.add(label)
```

```plaintext
    (add bin label)
```
Generic functions

\[(\text{add bin label})\]
\[\text{generic function}\]

\[> (\text{generic-function-methods add})\]
\[\text{(#<<method> (\text{gst-bin} . \text{top})}
\[301c3bb0>\]
\[\text{#<<method> (\text{gtk-container} \text{gtk-widget})}
\[301c3c00>\]
\[\ldots\)\]
Instantiating objects

(make makes objects
 (make <gtk-window>))

(make <gtk-window> :type 'toplevel)

Keyword argument sets GObject property
(define b
  (make <gtk-button>
    :label "click me")
)

(connect b
  'clicked
  (lambda (b)
    (display "Hi FOSDEM!")))
(connect b 'clicked (let ((i 0)) (lambda (b) (set! i (+ i 1)) (display "Clicked ") (display i) (display " times"))))
(Parenthetical note)

(connect
  b
  'clicked
  (let ((i 0))
  (lambda (b)
    (set! i (+ i 1))
    (display "Clicked ")
    (display i)
    (display " times"))))
Deriving classes

(define-class <my-widget>
  (<gtk-vbox>)
  (my-slot
    :init-keyword :my-slot))
GObject properties & signals

(define-class <my-widget>
  (<gtk-vbox>)
  (my-slot
   :init-keyword :my-slot
   :gparam
   `(~<gparam-boolean>
     :default-value #t
     :flags (read write)))
  :gsignal `(my-signal #f)
Flags, enums, symbols

Enum values normally given as symbols; flags as lists of symbols

(make <gtk-window> :type 'toplevel)
(make <gtk-window-type> :value 'toplevel)
(make <gtk-widget-widget-flags> :value '(visible sensitive))
Other niceties

GNOME-VFS exposes Scheme ports

GdkPosition is a pair of ints

GConf and D-BUS type systems wrapped fairly transparently

....
Caveat hacker

Class vmethods not wrapped yet

Other class library↔Scheme mappings are possible: c.f. kawa
Which Scheme?
Guile Scheme

There are many Scheme implementations

I chose Guile as my Scheme for irrational reasons
Why Guile? (Rational reasons)
Excellent for extending C apps
Multithreaded
Widely available and deployed
Portable
Guile shortcomings

Interpreted; does not do native compilation (yet?)

Not much interest from Scheme community

Slow development, small dev team
Guile versions

1.6 is most common

1.8 is the current stable series, adds multithreading

CVS seeing evolutionary improvements
History & Status
Long gestation period

Guile was first serious language binding to GTK+, in 1997

Written by Marius Vollmer

Successor of that code is guile-gtk 1.2
Latter-day saints

libgobject begat guile-gobject, in 2001

Martin Baulig died in childbirth

Guile-gobject incubated a couple years with Ariel Rios
Metamorphosis

In 2003 a hapless hacker wanted to make a synthesizer with GStreamer and Scheme

4 years later I'm still here

I've been playing this song for 20 minutes now
I could play it for 20 more
I'm not proud
Or tired
guile-gnome-platform

- glib
-atk
-pango
-gtk
-libgnomeui
-libgnomecanvas

libgnome
libglade
gnome-vfs
gconf
corba

Wraps GNOME 2.8 libs
Currently updating to 2.16
Other Guile-GNOME bindings

Hildon: hildon osso

GStreamer

GtkSourceView

Attic: evolution-data-server, libwnck, libgda, dbus, panel-applet
about the binding
Naming conventions

GTK+ module: ( gnome gtk )

GtkWidget class: < gtk-widget >

gtk_widget_show

function: gtk-widget-show

method: show
How Guile-GNOME binds

Start with a .defs file
S-expression format also used by pygtk

Build G-Wrap objects

Tell G-Wrap to make C

Compile the C
Modular bindings

Bindings can depend on other bindings

Interdependent bindings can be built together, like g-g-platform

Can build apart, like gstreamer, or guile-gnome-glib minimal package
G-Wrap

http://www.nongnu.org/g-wrap

Bindings generator, C to Guile

Handles function and type definitions

Uses libffi to call functions
Wrapping GObject

Very clean and open, a bit esoteric
Extensible

New types can be added

Guile-GStreamer extends core with new fundamental type, GstMiniObject

G-Wrap extensible also, new wrapping behaviors possible
Challenges

We hit unoptimized paths in Guile's OO system

Solution: delay class, method creation

Compilation would be the big win
Stability

Anticipated need for API change

Parallel-installable API versions

(use-modules (gnome-0))

0 is unstable; 2.16.0 will bump API version to stable 1
Distro penetration

Only Debian, currently

Difficult because of G-Wrap, which has its own unhappy history
Example apps

Not many

Maemo/N770 vote counter
Photo uploading/scaling prog
GNOME speed reading trainer
Unfinished synthesizer
Other in-house apps
Documentation

Only very introductory docs :(

Comprehensive API docs difficult

GObject introspection will help?
demo?
The Future
The future
Go stable
Fire and motion: track GNOME
Expand bindings
Scratch an itch
Conclusion

Because it's fun!

Questions?

www.gnu.org/software/guile-gnome