The User in the Loop

Extensibility, Guile, and GNU
GNU Hackers Meeting 2011

Andy Wingo

0.1 Greetings!

Andy Wingo

Guile co-maintainer, along with Ludovic Courtès

0.2 Goal

Understand the problem

Guile is part of the solution

0.3 Agenda

Extensibility: Macro perspectives

E11y for users

E11y for GNU

Getting there

0.4 Why Are We Here?

To advance software freedom?

Is that it?

Is that the end goal?

Other things that people bring up: to foster a culture of sharing; to have fun; to make awesome things; ...

I'll focus on the last point.

0.5 Making the Right Thing

The MIT approach vs. 'worse is better'

Emacs Makes A Computational Space?

A common thread: Richard Gabriel

- Patterns of Software

ANSI Common Lisp, CLOS, the Gabriel benchmarks, an interesting OO database and IDE; now a poet.

Other Gabriel things: Lucid Emacs -> XEmacs (boo), League for Programming Freedom (tape spools, heh).

Note that here I don't argue one way or another on the worse-is-better question.

http://dreamsongs.com/NewFiles/Timrep.pdf

http://dreamsongs.com/WorseIsBetter.html

http://dreamsongs.com/Files/PatternsOfSoftware.pdf

0.6 The Quality Without a Name

Thesis: Some places just feel right

Christoper Alexander: This feeling is ‘living structure’

Architectural patterns help produce that feeling

What about software?

Gabriel started mentioning Alexander in the Journal of Object-Oriented Programming.

Gemma et al did so before, but Gabriel’s work is deeper.

Federico Mena Quintero’s recent GUADEC talk, “Software with the quality without a name” `http://people.gnome.org/~federico/misc/software-with-qwan.pdf`

0.7 Architect vs World

Some patterns from Alexander’s ‘A Pattern Language’:

- Your own home
- Site repair
- Self-governing workshops

Is possible for our programs to make our users whole, to satisfy their needs, without the possibility of of modification, of change, of extension?

Your own home: people are divided from themselves if they do not own the place the live in. Renting is harmful in the long term.

Site repair: Buildings should be extensible by their users, on-site. Tools should be on-site.

Self-governing workspaces: A social argument that groups of working people should be able to make the important decisions affecting them in a democratic fashion. Note, this implies the ability to extend their buildings!

Alexander argues for the moral component of architecture: it should make people feel more whole.

Extensibility, piecemeal growth, is the key to long term development.

0.8 More Than Nice-to-Have

E11y is fundamental to human agency and happiness

Moglen: ‘Software is the steel of the 21st century’

Incremental change, qualitative change

{Counter,}example: GNOME 3.0

Software as a means to Real-World agency: Tunisia and all the rest

Witness GNOME 3 feedback. A minority is really irritated at GNOME, and it is a clear case of lost agency. GNOME 3.2 will focus on extensions.

0.9 Building Materials

Le Corbusier's concrete; GNU's C

Gabriel: 'The good news is that in 1995 we will have a good operating system and programming language; the bad news is that they will be Unix and C++.'

Refuge de l'Armée du salut, 12 rue Cantagruel

Pour l'architecte moderniste, le logement devait être une "machine à habiter" universelle, transposable partout quel que soit l'environnement ou le climat. Ainsi, les vitres ne s'ouvrent pas en raison de l'installation prévue d'air conditionné. Mais celle-ci ne fonctionnant pas...

Le Corbusier considered this building to be a universal "living machine", transplantable anywhere in the world, regardless of the environment or climate. To this end, the windows were not designed to open, because of the pending installation of an air conditioning system. But this system didn't work, so le Corbusier was obliged to ...

0.10 Guile

'This is a song about Alice. Remember Alice?'

Guile: Practical e11y for GNU

3 models:

- Embedding (within a program)
- Extending (outside a program)
- Scripting (the space between programs)

The reference is to Arlo Guthrie's tune, 'Alice's Restaurant'.

Glyph Lefkowitz makes the embedding vs extending argument here: `http://twistedmatrix.com/users/glyph`

0.11 Embedding

App links to lib$lang

App provides extension points: functions and data types, mostly

0.12 Guile Facilitates Embedding

Libguile: A lovely C interface

Widely deployed

Forgiving of mistakes

- Garbage collection
- Functional programming

Some tool support to help create correct extensions

No static typing though

0.13 Embedding Tips

Think about user experience

Debugging: usually printf-based

Provide a console if you can

Documentation is essential

I started this way

Embedding a language in your program is a tricky thing, because the environment is strange. It's not the normal environment that $language programmers are used to.

Providing a console is tricky because a REPL is its own main loop, and your app probably already has a main loop. If your app is mostly threadsafe you can use Guile's –listen facility, and run the REPL in its own thread.

Regarding tool support, I refer to the automatic compilation with compile warnings.

True fact: I did start with Guile by extending Gnucash to add a different kind of report. I did so from my house in northern Namibia, where I was living at the time without an internet connection. Good times!

0.14 Embedding Is Inferior

Embedded extensions compose poorly

Embedding privileges C over $lang

The developer experience is poor

Tools support is poor

Indeed, the user's extensions are inferior to the program itself, in control-flow.

0.15 Beyond Embedding: Extension

Turn your app into a library

Bind that library in $lang

The downside to this strategy is that you need to provide some degree of stability for your app, and that's not easy. Making a library is harder than making an app.

0.16 The GNU Extension Language

Guile is a fantastic extension language.

0.17 Practical Extensibility

Guile over C(oncrete):

- Language safety
- Source code availability
- Online compiler
- Online debugger
- Online apropos / meta-dot
- Multi-paradigm: Functional, OOP, message-passing, ...

Easy e11y == more extensions

More extensions will translate to happier users. Firefox is a good example here.

To maximize these advantages, you should write as much in Scheme as possible.

0.18 RDD

REPL-driven development

```
guile> (import (my-app))
```

Put the user in the [read-eval-print] loop

REPL provides discoverability and hackability

As my friend Jao says, when it comes to debugging, you can be a mortician or you can be a physician. Core dumps are corpses. REPLs live.

0.19 Guile: A Practical Scheme

Draw with Cairo, make UIs with GTK+ or Clutter, fetch and serve HTTP with the built-in web modules, parse and generate XML with SXML, access RPC services with Guile-RPC, do so securely with GnuTLS, access databases with Guile-DBI, process mail with mailutils, build distributed job servers and workers with ZeroMQ, make new bindings with the dynamic FFI, extend your way to a new language with modules and macros, write powerful servers with the POSIX bindings and native pthreads, load your extensions quickly with the bytecode VM, write your own control constructs with delimited continuations, do all this from Emacs without restarting Guile...

Scheme as a standard lacks libraries. Guile, on the other hand, does not.

0.20 Distributed Extensibility

Where are the programs we are extending?

- on traditional computers
- on devices
- on the web

Practical extensibility must reach these all platforms

Remote development and debugging in Emacs: Geiser

Web hacking: Guile's web modules

0.21 Extensions and GNU

(Exhale.)

(Inhale.)

0.22 Extensions and GNU

E11y: for whom? Just for the the user-programmer, or to also share?

E11y begins with the user-programmer

GNU can facilitate sharing; sharing can build GNU

Again, Firefox is a great example of sharing extensions.

0.23 Building the Guild Hall

A CPAN for Guile

Port of Debian's APT

Use it as an extensions archive for your program

Your extensions are guild packages

0.24 Extensions Are Gateway Drugs

Users are already sold on ideology, now hook them in with practical programming environments

The GPL is all we need; no copyright assignment!

0.25 But What About Scripting?

Scripting: The space between applications

All about protocols

Canonical example: SH and POSIX: Pipes between processes

0.26 Challenges & Opportunities

Scripting languages tend to not be good general-purpose languages (Tcl)

Platforms change, scripting languages (often) don't

Decline of Tcl, Perl

What is the current 'space between programs'? What will it be in the future?

Guile: an adaptable general-purpose language

- Modules, macros, syntax, semantics

0.27 Sites of the Future

In the future things will be distributed

We have a chance to make it

Guile can play an important part

Lots of tinkering, engineering, and science to do

0.28 Hacker vs User?

'This monument must be preserved against the passing of time.'

–Ricardo Bofill, architect of Barcelona's new airport terminal

0.29 The User in the Loop

E11y builds the living GNU project – not the monument.

Strengthening programs (centers) while organically knitting cohesive, living common spaces.

Whenever I'm in this terminal, I feel like I'm on the set of Tati's Playtime.

Here in Paris' 13th...

0.30 Final words

Christopher Alexander speaking to computer scientists at OOPSLA 1996

How the hell do you produce this living structure? What do you have to do to actually produce it? You can clumsily try to find your way towards it in a particular case. But, in general, what are the rules of its production? The answer is fascinating. It turns out that these living structures can only be produced by an unfolding wholeness. That is, there is a condition in which you have space in a certain state. You operate on it through things that I have come to call ‘structure-preserving transformations,’ maintaining the whole at each step, but gradually introducing differentiations one after the other. And if these transformations are truly structure-preserving and structure-enhancing, then you will come out at the end with living structure. Just think of an acorn becoming an oak. The end result is very different from the start point but happens in a smooth unfolding way in which each step clearly emanates from the previous one.

–IEEE Software volume 16, No. 5, September/October 1999

0.31 fin.

- `http://gnu.org/s/guile/`
- `http://wingolog.org/`