Using Potluck to break the Guile stalemate

Alex Sassmannshausen

September 18, 2017

Outline

Alex Sassmannshausen

Guix Potluck: what is it?

- Over the period of March and April, Andy Wingo wrote an extension to Guix that was inspired by discussions around Guix Channels during Fosdem.
- The resulting code, according to email discussions on the guix-devel mailing list, fulfills the following roles:
  - provide a more decentralized way of developing "Guix packages"
  - provide a complement to the current set
  - provide a sloppy way to share projects, via Guix, that are still prototypes / work in progress
  - the explicit stated collection would be unplanned, would contain duplicates, and be entirely the result of user contributions

Small detour: problem space.

- Guile has hitherto not had a great way to share user created libraries, modules & applications.
- Guix could provide an easy way to solve this problem.

Scenario: I have a small Perl library

- It follows standard Perl conventions
- Guix has a Perl build system that, in this case, should just do the right thing.
- I have never worked with Guix, let alone Lisp.
- I want to leverage the awesome of Guix.
- I've installed Guix! :-DDDDDDDDDDDDD
- Now, all I have in front of me is an empty text file...
  ^—- This is the problem I'm concerned about

After Ludo's talk: let's revisit the Guix package specification

```
(define-module (gnu packages hello) #:use-module (guix packages)
#:use-module (guix download) #:use-module (guix build-system
gnu) #:use-module (guix licenses) #:use-module (gnu packages
gawk))
(define-public hello (package (name "hello") (version "2.10")
(source (origin (method url-fetch) (uri (string-append
"mirror://gnu/hello/hello-" version ".tar.gz")) (sha256 (base32
"0ssi1wpaf7plaswqqjwigppsg5fyh99vdlb9kzl7c9lng89ndq1i"))))
(build-system gnu-build-system) (arguments '(#:configure-flags
'("–enable-silent-rules"))) (inputs `(("gawk" ,gawk))) (synopsis
"Hello, GNU world: An example GNU package") (description
"Guess what GNU Hello prints!") (home-page
"http://www.gnu.org/software/hello/") (license gpl3+)))
```

Looking through fresh eyes. . . (perhaps)

(define-module (gnu packages hello) \ #:use-module (guix packages) \ #:use-module (guix download) \ Requires knowledge of Guix module structure #:use-module (guix build-system gnu) / #:use-module (guix licenses) / #:use-module (gnu packages gawk)) /
(define-public hello (package <—— Package record is nice to read, tricky to write in the beginning (name "hello") (version "2.10") (source (origin \ What is an origin? (method url-fetch) \ (uri (string-append "mirror://gnu/hello/hello-" version \ ".tar.gz")) | (sha256 | (base32 | "0ssi1wpaf7plaswqqjwigppsg5fyh99vdlb9kzl7c9lng89ndq1i")))) | How do I generate this? (build-system gnu-build-system) - Which one should i use? (arguments '(#:configure-flags '("–enable-silent-rules"))) (inputs `(("gawk" ,gawk))) - Linked to module structure (synopsis "Hello, GNU world: An example GNU package") (description "Guess what GNU Hello prints!") (home-page "`http://www.gnu.org/software/hello/`") (license gpl3+)))

Is Guix Potluck a solution?

Let's investigate!
Let's take it for a ride!

Let's do it!

My WIP: Guile Config (changed scenario: simple Guile module)
$ cd /to/project/dir $ guix potluck init
`https://gitlab.com/guile-projects/guile-config` \
–autotools –license=gpl3+ $ guix build –file=guix-potluck/config $
guix potluck update
`https://gitlab.com/guile-projects/guile-config`

Well that's nice and easy!

- OK:
  - we don't have a guile build system.
  - The GNU build system is bewildering as a new comer!
- Let's ignore those problems for now.
- Main points:
  - we did not start with an empty page!
  - we did not need to know where inputs are located!
  - we have access to licenses & build-systems!
  - most of the package is written for us!

Change of scene: How is Potluck implemented?

There are two parts.

Guix client side

- A WIP branch in the Guix repository
- Implements a subcommand, ‘guix potluck‘
  - Which uses Guix modules to allow the module contributor to generate a “potluck package“ — a simplified guix package.
  - Which can also "install" a repository containing a “potluck package“ in a potluck server.

Server side

- A guile server which can be passed a reference to a repository containing a “potluck package“.
- It tries to build this package
- Then adds it to a local ”source” repository: a giant git repository of user contributed ”potluck packages”
- Then transforms the package into a ”guix package”, which is then added into a giant git repository of compiled ”potluck packages”: a repository of ”guix packages”.

How it can be used

There are two user roles

The module producer

- The person who wishes to contribute a module to the potluck repository.
- They use the potluck subcommand to generate and upload a potluck package.

The module user

- The person who wishes to use packages that were submitted to the potluck repository.
- They clone the potluck guix package repository inside a directory made accessible to Guix using GUIX$_{\text{PACKAGEPATH}}$.
- They use the normal guix commands to install any packages from the potluck repository.

How can this benefit Guile [some background!]

Guile has historically not had an effective way of distributing user contributed code.
Previous attempts have revolved around:

- Guile-Lib
- Guile-Core
- Dorodango/Guildhall

The first doesn't scale, the second would accept only very high quality code. The third never took off.

Guix' impact on Guile

Guix has caused the Guile community to grow. I would argue that we have more Guile packages in Guix than in any previous attempt at providing a Guile package manager.
How do we: a) make it easier for people to package Guile packages? b) accelerate the process by which we make Guile packages discoverable through guix?
Some conclusions so far in general terms!!!!!!111111

Potluck as a solution to (a)

Potluck provides a semi-interactive script to generate guix package like templates.
They can be populated fairly easily by the module maintainer, making generating Guix compatible packages easier for non-Guix users.

Potluck as a solution to (b)

Potluck packages generated can be shared instantly, without going through a QA process.
They can then be discovered by anyone doing a git clone of the resulting repository: the packages become searchable by Guix.

Potluck as staging for Guix

Once a package attains maturity, migrating from Potluck to Guix should be easy: the hard work has already been done.

What makes Potluck nice & easy to use?

- Potluck init avoids the “empty page“ situation.
- The Potluck package specification avoids the need to specify exact package variables, freeing the user from having to know the Guix packages file structure.
- Potluck update makes making a package discoverable easy.

Potluck as the solution to the Guile stalemate?

- Potluck feels lovely to use:
  - init feels like it provides a clear user journey with instructions
  - upload has a very docker registry like feel about it in terms
  - I would argue the UI provides a friendly entry to the Guix adventure.
  - The potluck initiative is worth pursuing already because of the above factors.

The End?

Well,... maybe.
There are other elements to this discussion (e.g. Guix Channels).
I will ignore those for now.
My main motivation: make Guile/Guix contribution easier.
What follows here are some loose thoughts, not yet backed by code!
(Which means: what Andy created is already infinitely more valuable than what I recommend, for lack of code!).

What could be improvements to Potluck from the beginner's perspective?

- Users should be exposed to full package references as soon as possible, so the start getting familiar with it:
  - When learning by reading and doing, it is better to study the guix package specification directly, rather than yet another format.
  - We can reinforce the already existing custom of storing a guix.scm file inside project repositories.
  - We make the guix package definition, which would eventually be used for integration into Guix immediately available to the package producer, thuse encouraging a transition from "Guix user" to "Guix contributor".
- There is a niche for a simplified package spec for convention compliant, first-time Guix package contributions.

Proposal: extend the init command!

- allow for specifying “inputs“, “synopsis“ & ”description” on the command-line.
- allow for different output types: JSON, YaML, messy-Wisp, messy-Guile.
  - All these would allow input etc. references as strings, and would automatically generate the source specifications.
- provide an importer from JSON, YaML & messy-Wisp & messy-Guile packages specifications to Guile package specifications (messy importer).
- The result would be a command-line utility that makes it easy to generate “guix.scm“ files in any project repo.

Proposal: make the Potluck server work with package specifications.

- We simplify the server, maintain its Continuous Integration capabilities & its decentralised discoverability
- We enhance its supporting role for the Guix package manager by encouraging & exposing direct Guix packages.
- We encourage migration to Guix through & Guile by encouraging distributed projects to provide a guix.scm file.

NOTE: we still need to generate full Scheme modules on the server, to work nicely with GUIX$_{\text{PACKAGEPATH}}$. The local guix.scm file should not be a module definition: it should just be a file returning a package object.

Proposed workflow

- Within project repo:
  ;; guide through initial messy creation $ guix spring init –output=json > messy-guix.scm
  ;; test build from messy file: trial & error $ guix spring build
  ;; convert messy to full guix package specification $ guix spring import messy-guix.scm > guix.scm
  ;; upload/update on discoverable registry $ guix spring update guix.scm

What do we gain?

- Beginners have a command that sets them on the right path
- Newbies can "fill in blanks" in command or simplified template without worrying about stringent Guix "type checking", until their messy definition works
- Travellers from non lisp-y lands can work in their preferred declarative syntax, until it isn't good enough any more
- We encourage adoption of a standard Guix based development workflow
- We make dev projects with a Guix workflow "discoverable"

The End

- This is a topic with plenty of scope of discussion!
- What do you think?
- Thank you.

ø/
[Thank you to Ludo & all Guix devs; Thank you to Andy for Potluck!]

Plan of action

- Inherit GNU build system with autoreconf option
- New build system: Guile
- Extend Init:
  - parse input, synopsis, description arguments
  - print to simple-guile
- Add transform:
  - parse simple-guile, expand to package
  - attempt to build
  - on success, print guix.scm
  - else error meaningfully