The Library Infrastructure Project

Isaac Jones
HIM: Saturday 29 August, 2003
For the use of his lazer pointer

Lambda’s Namesake:
“Languages flourish when libraries are plentiful, reliable, and well documented.” – SPJ

Currently, there is no great way for tool authors to contribute and widely distribute their libraries and tools except to have them included with the implementations.

BUT... This is a strain on the implementation & library authors.

Let's give library & tool authors a way to “contribute” their software
Issues Facing 3rd Party Tool Authors

- Difficult to distribute binary Haskell libraries
- ... so the end user must build (and rebuild) all the libraries on their own system
- ... but there is no standard build system
- ... all of which make it hard to build Debian packages (for instance)
Issues Facing 3rd Party Tool Authors

- Several Haskell implementations which treat “packages” differently (different binary formats, different means of collecting packages)
- Language extensions and supporting libraries are a moving target (and oh, so tempting), causing the bitrot of tools that aren’t actively maintained
- No way to express dependency on particular libraries, compilers, or versions thereof (job of the packaging system?)
- No central repository for packages / libraries
Why Should We Solve This

Its all about the community...

- Help operating system packagers build packages (Debian, RPM, etc) to keep users happy
- Give library authors ways to contribute their libraries in a “Bazaar” style
- Help the community feel they “own” the open-source projects and give them a common set of tools to maintain them, as Debian does.
- In Debian, everyone knows how to: file bugs, download & build source, submit patches, announce new projects, ask for help maintaining tools, flame
What a Solution Might Look Like

- A nice build system with which a library author can build binary versions for a variety of architectures and implementations (in practice, this is a very large number of binaries)

- A repository where the author can announce or upload their tool
We’re Already on Our Way

Building
- “FPTools” make-based system. Point of contact: Alastair Reid
- Yale’s make-based system. Point of contact: Henrik Nilsson
- HMake Haskell-based system. Point of contact: Malcolm Wallace

Announcing
- Haskell mailing lists
- The haskell.org web page and Wiki

These are a big step forward! Keep up the good work!
A Haskell-Based System

I propose a Haskell-based build system which performs the following tasks:

- Compiles or prepares Haskell libraries and tools
  - By reusing code from hmake to build directly or
  - By calling through to a make-based system
- Installs Haskell libraries and tools
- Tracks metadata about installed packages and Haskell implementations (a new packaging system)
Taking a page from Python’s book, each distributed library or tool (except for the compilers) comes with a Haskell program, Setup.hs which provides standard targets to wrap other build systems, or builds the packages itself.
Why Haskell-Based?

- The one thing that all the systems of interest have in common: Haskell
- Side-effect of improving the libraries needed for common scripting tasks (lets steal some of the market from Python)
- Haskell beats Make for abstraction and reuse
- Reuse: Each piece of the project (Building, Installing, and Packaging) can be leveraged elsewhere if we make them into libraries
- “Eat your own dogfood” is a good policy
Outline

- Building: Strategies for build systems
- Installing: Setup.hs scripts to build and install Haskell libraries and Tools
- Packaging: How we can store and leverage what we know when we know it
- Tool Support: Tools which could be layered on top of a module
Module Hierarchy for Distribution

- Distribution.Build
  - dependencies :: [Package] -> Graph Packages
  - build :: Package -> Compiler -> IO ()

- Distribution.Package
  - data Package {...}
  - getSystemConfig :: IO SystemConfig

- Distribution.Installation
  - install :: Package -> Compiler -> IO ()
  - register :: Package -> IO ()
  - sourceDist :: Package -> IO ()
  - bdist_debian :: Package -> IO ()
System Overview

- /etc/haskell/packages.conf,
  ~/.haskell.conf

- Code Browser, Debugger, etc

- import

- fptools

- wraps

- myPackage

- compile

- haskell-pkg

- import

- Distribution.Package

- import

- Distribution.Install

- import

- setup.hs

- compile

- import

- myPackage

- compile

- ghc

- compile

- nhc

- compile

- hugs

- compile
That is the end of the overview. At this point, I hope you understand:

- The motivation for this project
- Some implementation ideas for this project
- Who would use it and how
Why building is hard:

- Several very different Haskell implementations
- A variety of operating systems and hardware architectures
- Lots of preprocessors and foreign libraries
Building: Basic strategy

For simple tools like Haskell modules, leverage HMake’s abilities and create a Haskell-based system (which may evolve to do more complex tasks.)

Complex systems can use “fptools” or Yale’s Make-based system, or their own build system.

All systems will be wrapped in a common veneer (Haskell program) so they look the same to the average user, and to layered tools (like Debian).
**Tasks for Distribution.Build**

API For:
- Compiling for a particular Implementation (like hmake)
- Compiling for all installed implementations
- Abstracting some implementation-specific flags

Can be used for:
- Asking compilers to build Haskell code
- Dealing with some preprocessors
- Building higher-level tools on top (later slide)
- Recompiling when a new Implementation is installed
- Implementing a generic `/usr/bin/haskell` (like hi)
The main feature of the Installation Module is a script which imports Distribution.Build, and interfaces with the packaging mechanisms discussed below.
Setup.hs Strategies

- `#!/usr/bin/env haskell` (something haskell-interactive inspired?)
- Import `Distribution.{Build,Install,Package}` which can take care of major tasks
- `main = distributionMain Package{...insert package meta info here...}`
- Standard libraries may need richer OS operations
- ...but this is a good thing, it can help Haskell to get more market share in the scripting area
Command-line arguments

./Setup.hs

- install-\{default, all, nhc, ghc, hugs\}
- build-\{default, all, nhc, ghc, hugs\}
- bdist-\{deb, rpm\}
- sdist – makes a tarball on unix
Example Setup Program

#!/usr/bin/env haskell
import DistUtils.Core
import DistUtils.ToolInfo

toolInfo = (basicPackage (OtherTool "HUnit")
              (Version 1 0 0))

    {haskellSources=[
      "HUnitLang98.lhs","HUnitLangExc.lhs",
      "Terminal.lhs", "HUnitTest98.lhs", ...],
    docs = ["Example.hs", "Guide.html", ...]}

main = distUtilsMain toolInfo
Much of this becomes easier with a more generic package system, which has benefits outside of this project.
Jobs of a Packaging System

- Track which Haskell Implementations are installed
- Track which preprocessors are installed
- Track which libraries and tools are installed
- Find the source code for modules when needed
Packaging

Diagram:

- Haskell program
- Hugs
- GHC
- NHC
- Distribution.Package
- packages.conf
- haskell-pkg
- Haskell Tool
- Non-Haskell Tool
- make-based system

Compilation Request

Package info
Think of debian/control combined with Package.conf

- *Things the build system cares about:* Source Files, Build Flags, Build Dependencies
- *Things the build system doesn’t care about:* Name, Dependencies, Description, Version, License Information, Home Page
Tools layered on Packaging System

- Build & Install system
- Debuggers which need to instrument code
- Source code browsers
- The Glorious Glasgow Haskell Compiler Source Code Deleter (find other versions of software and “repair” any possible type errors)
Layered Tools

- Creating distribution packages (Debian, FreeBSD, Windows, etc.)
- Web database of Haskell tools
- Installation (usually already there)
- Removal (often not there)
- Package registering and rebuilding
- Downloading and installing dependencies (job of parent system?)
- Verifying authenticity of packages (via cryptographic signature)
Conclusions & Directions

I have implemented a prototype (which interfaces with Debian’s build system), but it’s blocked on a packaging system.

After HIM I will write a new proposal and try to create consensus.

But where do you think I should direct my attention (make-based system? CPAN-type archive? Distribution module?)

My opinion: Packaging decisions, then Distribution module.
(Assuming that we haven’t run overtime and everyone is ready to go to lunch)