ghc-mod

Making Haskell development even more fun

Daniel Gröber Kazu Yamamoto

August 31, 2015

Outline

1. Motivation
   - What is it?
   - Why work on it?
2. Implementation details
   - Current architecture
   - Redesigned architecture
3. The internship
   - What we have done so far
   - What is still to be done

Outline

1. Motivation
   - What is it?
   - Why work on it?
2. Implementation details
   - Current architecture
   - Redesigned architecture
3. The internship
   - What we have done so far
   - What is still to be done

What is ghc-mod ?

First some marketing blurb:

ghc-mod is a backend program for enhancing editors and other kinds of development environments with support for Haskell, a library for abstracting the black magic incantations required to use the API of the most popular Haskell compiler in various build environments and an Emacs Lisp frontend program to let users access it's features.

What does it do?

- `check` modules for compilation errors and warnings,
- get the inferred `type` of an expression in a module,
- `list` modules, compiler and `language flags`,
- `browse` symbols defined in modules,
- `find` which module a symbol was defined in,
- lookup `documentation` for a symbol or module
- and a bunch of more obscure things.

Outline

1. Motivation
   - What is it?
   - Why work on it?
2. Implementation details
   - Current architecture
   - Redesigned architecture
3. The internship
   - What we have done so far
   - What is still to be done

Why?

- It's actually rather popular:
- GitHub

- Hackage (Haskell package repository)

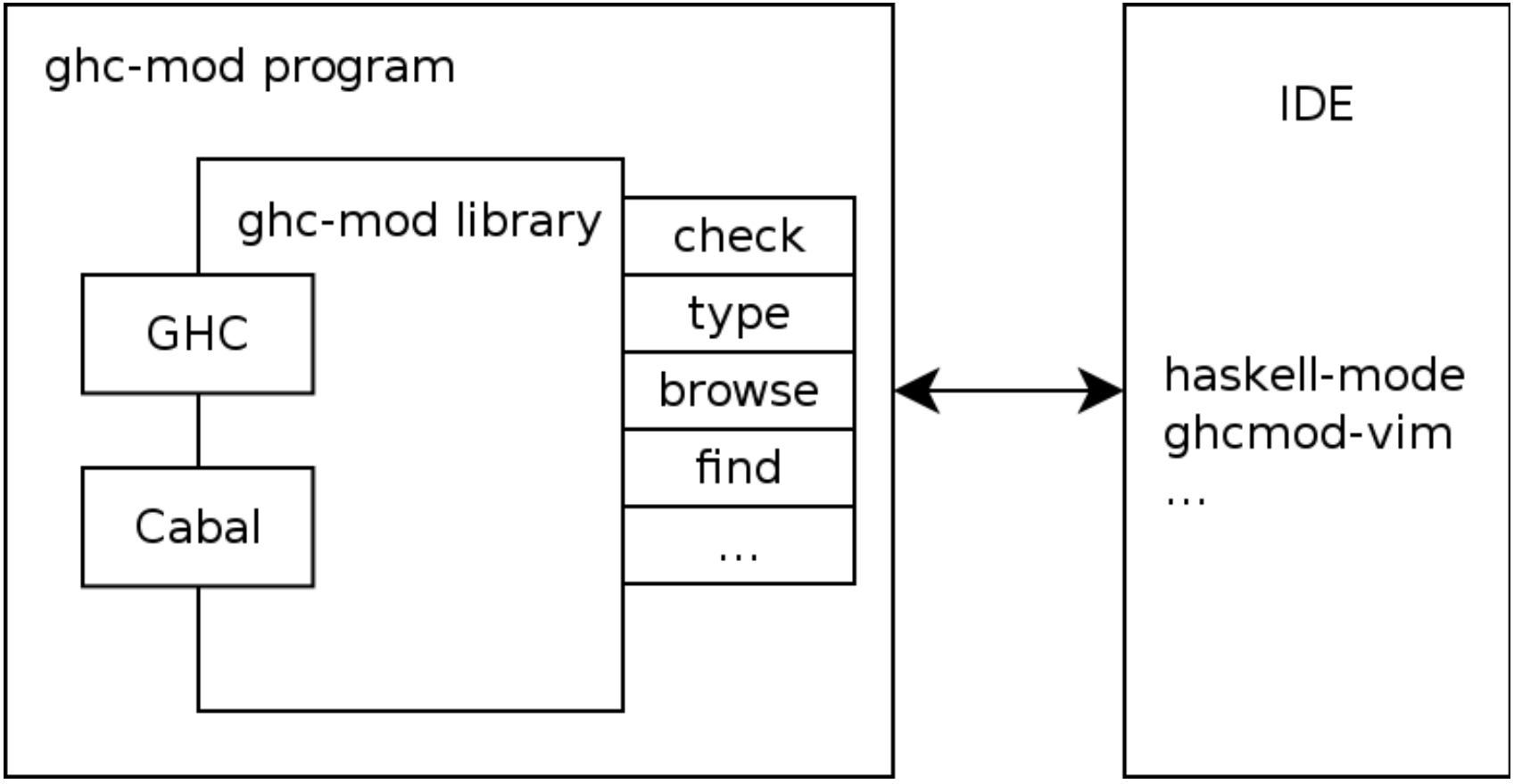
Downloads 47590 total (1588 in last 30 days)

- Also working with compilers is fun, right?

Outline

1. Motivation
   - What is it?
   - Why work on it?
2. Implementation details
   - Current architecture
   - Redesigned architecture
3. The internship
   - What we have done so far
   - What is still to be done

Current architecture



ghc-mod the Elisp program

- Extends haskell-mode to allow access to ghc-mod's features
- There really isn't much more to it than that

ghc-mod the program

- Development environment communicates with `ghc-mod` process
- Exists as a one-shot and long running process version
  - `ghc-mod` simple, doesn't have to worry about caching
  - `ghc-mod` "interactive" much more complex, needs to be very aware of changing environment and how that affects compiler internal caches
- interactive `ghc-mod` is generally much faster than `ghc-mod` at least for features that require compilation though

ghc-mod the library

- ghc-mod frontend programs use the library to implement all functionality
- Frontends are very thin wrappers around the library, all the intelligence is in there
- Primary entry point abstracts away environment setup and just gives the underlying tool a compiler session to work with
- Right now it's only of limited use for implementing new ghc-mod like tools on top of it and definitely needs a redesign (for v6.0 probably)
- Alan Zimmerman's Haskell Refactorer (HaRe) uses it for example

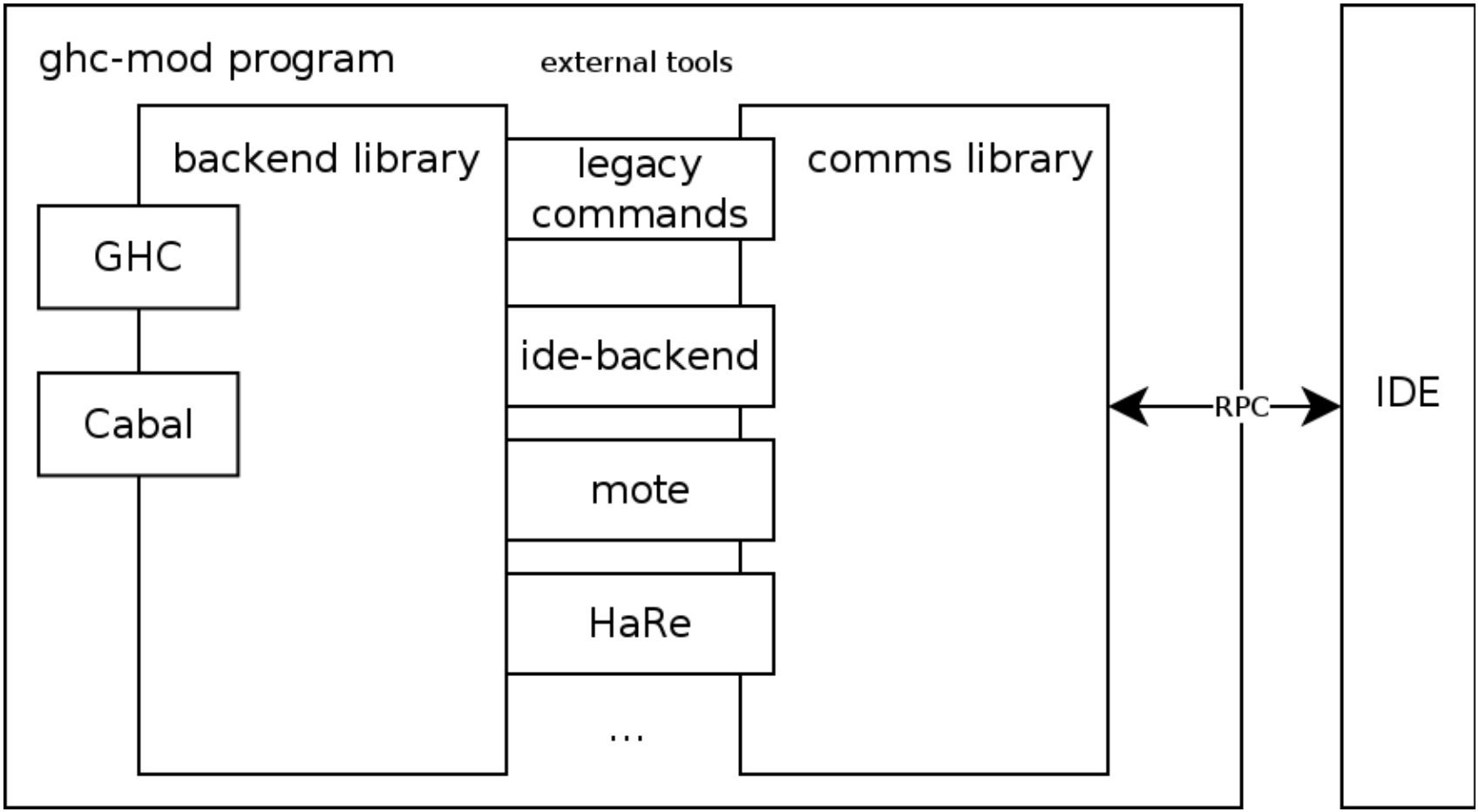
Problems

- Extending `ghc-mod` from the outside is hard to impossible
- External tools end up depending on ghc-mod making it difficult for us to make use of them
- This all just leads to fragmentation in the already fragmented Haskell Tooling Landscape
- one tool, `mote`, just ended up copy-pasting part of `ghc-mod`'s environment support code straight into it's codebase `-.-`
- development environments essentially need to support every tooling project themselves

Outline

1. Motivation
   - What is it?
   - Why work on it?
2. Implementation details
   - Current architecture
   - Redesigned architecture
3. The internship
   - What we have done so far
   - What is still to be done

Redesigned architecture



Redesigned architecture

- Factor out commands from library into a seperate package
- Refine the library so any tool can actually make use of it
- Design a communication library towards the development environment which provides some common ground for tools and frontend developers

Outline

1. Motivation
   - What is it?
   - Why work on it?
2. Implementation details
   - Current architecture
   - Redesigned architecture
3. The internship
   - What we have done so far
   - What is still to be done

Bitrot

- Cabal version 1.22 completely broke `ghc-mod's` hack'y way of getting information about the build system state
- To fix this (recurring) problem once and for all we had to completely re-design how we access Cabal's internal state
- Next GHC version 7.10 came along and also broke `ghc-mod`
- Adding support for the new compiler version was easy
- Cabal-1.22 support was however still blocking the release

Outline

1. Motivation
   - What is it?
   - Why work on it?
2. Implementation details
   - Current architecture
   - Redesigned architecture
3. The internship
   - What we have done so far
   - What is still to be done

TODO

- Essentially implement all of the architectural changes
- Support for implementing REPLs on top of ghc-mod
- Speed up `ghc-mod` program by adding network RPC support

Questions?