IDE Support in GHC

HIW 2017

Alan Zimmerman

Overview

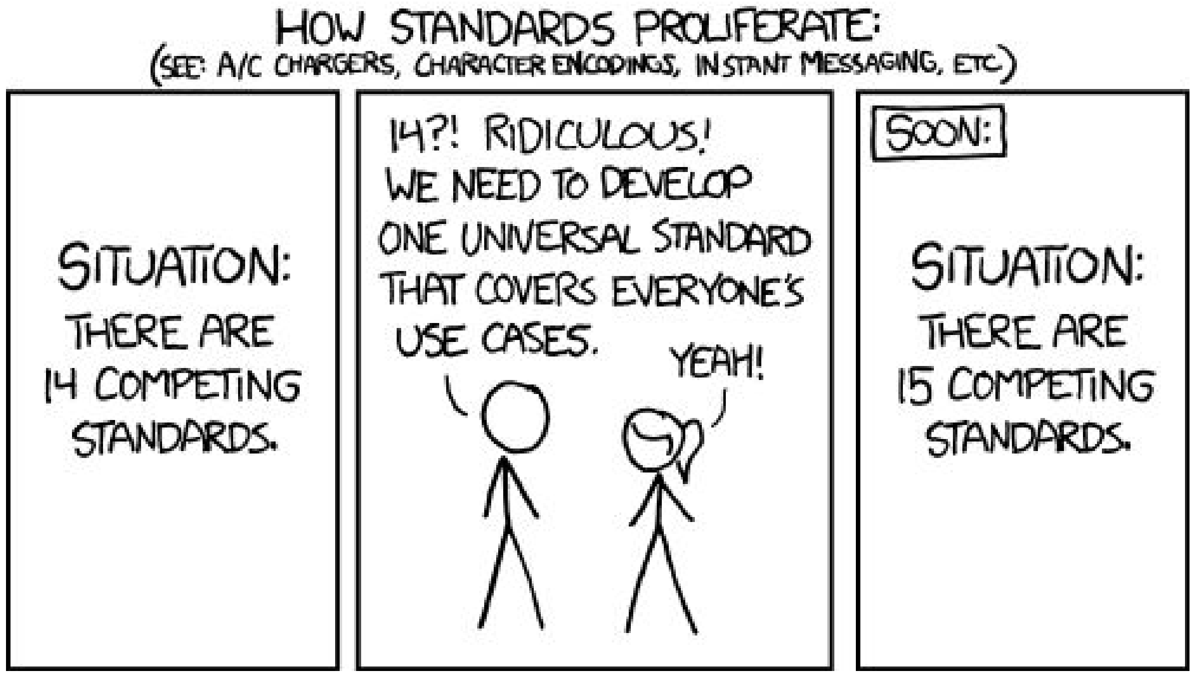
Historically IDE support has been very specific to the IDE

Pieces that need to be managed, for every tool

- IDE client(s)
- The compiler
- The context of the thing being developed

Alternative

- Standardise as much as possible


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Haskell IDE Engine

- Started in 2015 as a first attempt to unify the tool space
- Lacked focus, through not being targeted at a specific use case
- Did not have any devs on board focusing on actual IDE integration
- Chicken and egg situation

Language Server Protocol

- Microsoft initiative, developed against vscode
- Successor to OmniSharp protocol
- Client Server model, IDE is the client
- Typically supported in an IDE through a common library, and language-specific configurations

https://github.com/Microsoft/language-server-protocol

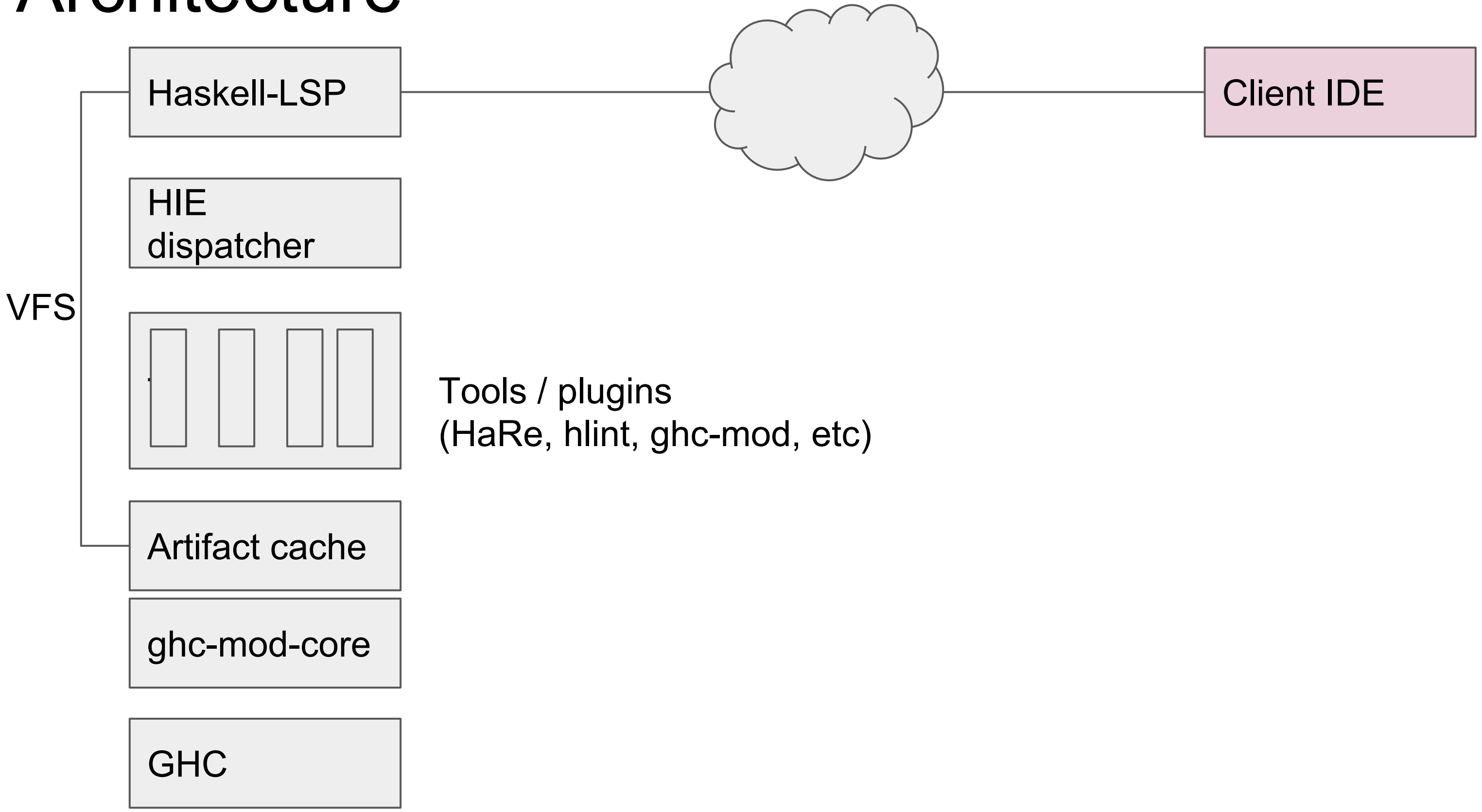
- 1,724 stars, 31 contributors
- Currently has 38 different language servers, 10 different IDEs
- See also http://langserver.org/, by SourceGraph

Haskell IDE Engine - Current status

- LSP support has been the focus of Zubin Duggal’s HSOC project
- We now have pretty solid Haskell support
- Tested against vscode and emacs clients
- Starting to be used in kakoune and neovim
- Preliminary experiments against atom and Yi
- Github repo has 553 stars, 29 contributors

Demo

Architecture



Virtual File System

- LSP can provide incremental changes as the file is edited
- haskell-lsp uses these to maintain a Yi.Rope structure for each open file
- When a plugin chooses, it can ask for the current file contents for processing.
- This is seamlessly supported in ghc-mod-core
- The caching layer stores this loaded module, so that all plugins can access it

Unaddressed issues / shortcomings

- No REPL support
- GHC 8.0.2 only

Long Term Goal

- Compiler as an API, similar to Roslyn (for C#) and Rust, Clang, ...
- Eventually a fully incremental demand driven compiler geared to tool use, computing artifacts lazily.
- Parsed Source AST should be fully round-trippable via the standard pretty printer.
    - Requires extra information storage, which will be built on the “Trees that Grow” facility being built into GHC by Shayan Najd.

First Steps

- We are already getting incremental updates via VFS
- Start with parser (work in progress)
- Based on “Efficient and Flexible Incremental Parsing” by Wagner and Graham (1998)
    - Modification of standard LALR(k) parser
    - Parses at the tree level, breaking the old tree down according to modification, and rebuilding it.
    - Can reuse unchanged tree fragments, supporting reuse of output from later phases too.

Current Status

- Happy modified to support the algorithm, using standard parser definition
- Beginning to push the changed source in a FingerTree through an incremental lexer into the parser.
- Using the approach from “Incremental Lexer for IDE” by Yuras Shumovich

Envisaged problem areas

- Incremental lexing, due to the stateful lexer for haskell.

Using haskell-ide-engine now

- https://github.com/haskell/haskell-ide-engine
- https://github.com/alanz/vscode-hie-server (or vscode marketplace)
- https://github.com/emacs-lsp/lsp-mode
- https://github.com/emacs-lsp/lsp-haskell

Other links of interest

- https://github.com/alanz/happy/tree/repetitive
- http://blog.haskell-exists.com/yuras/posts/incremental-lexer.html
- https://github.com/alanz/tide
- https://github.com/alanz/incremental-play
- https://github.com/alanz/haskell-lsp