Writing a Language Server in OCaml for Emacs, Fun, and Profit

Austin Theriault[1]

<2023-12-06 Wed>

[1] austin@semgrep.com

Outline

Who am I

- Software engineer @ Semgrep
- I work on our editor integrations
- I love working on programming languages, editors, cryptography

What is Semgrep

- Semgrep's core product is a SAST tool
- Can think of it as a security linter
- Supports 30+ languages
- Lots of customers, all using different IDEs

SAST

Static Application Security Testing

Demo

What is Semgrep

- Semgrep's core product is a SAST tool
- Can think of it as a security linter
- Supports 30+ languages
- Lots of customers, all using different IDEs

SAST

Static Application Security Testing

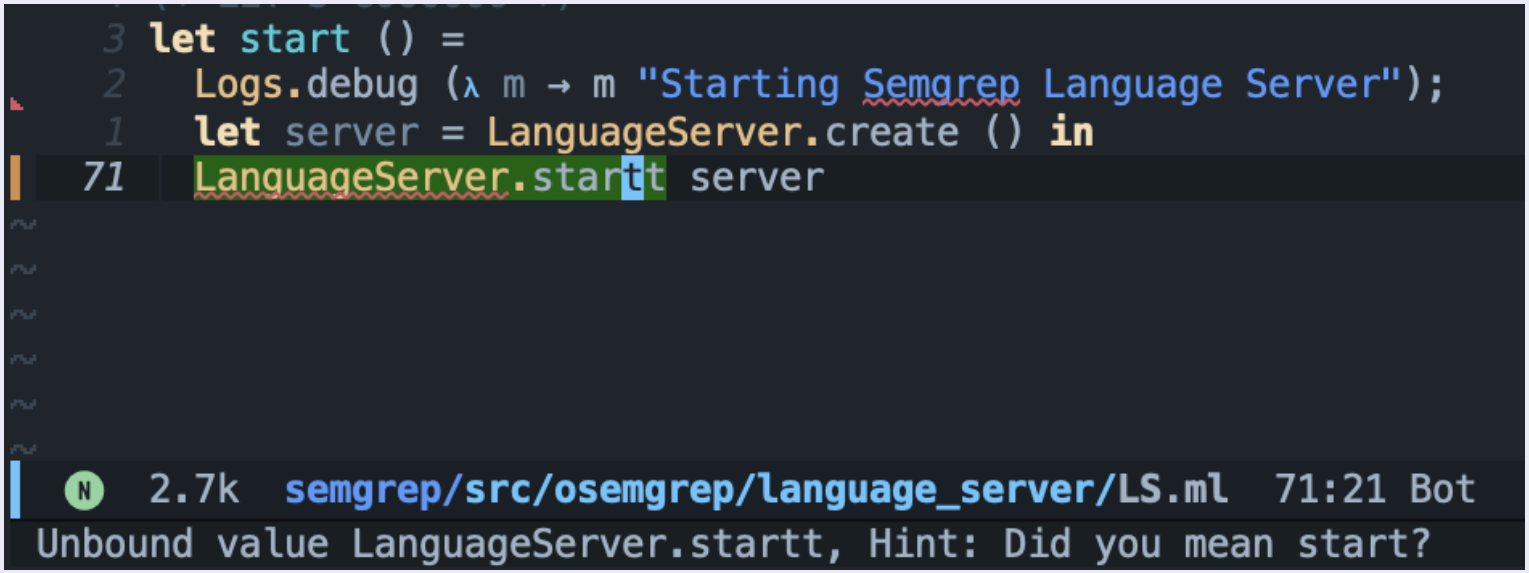
Demo

Product Purpose

Show security bugs as early as possible in the development cycle

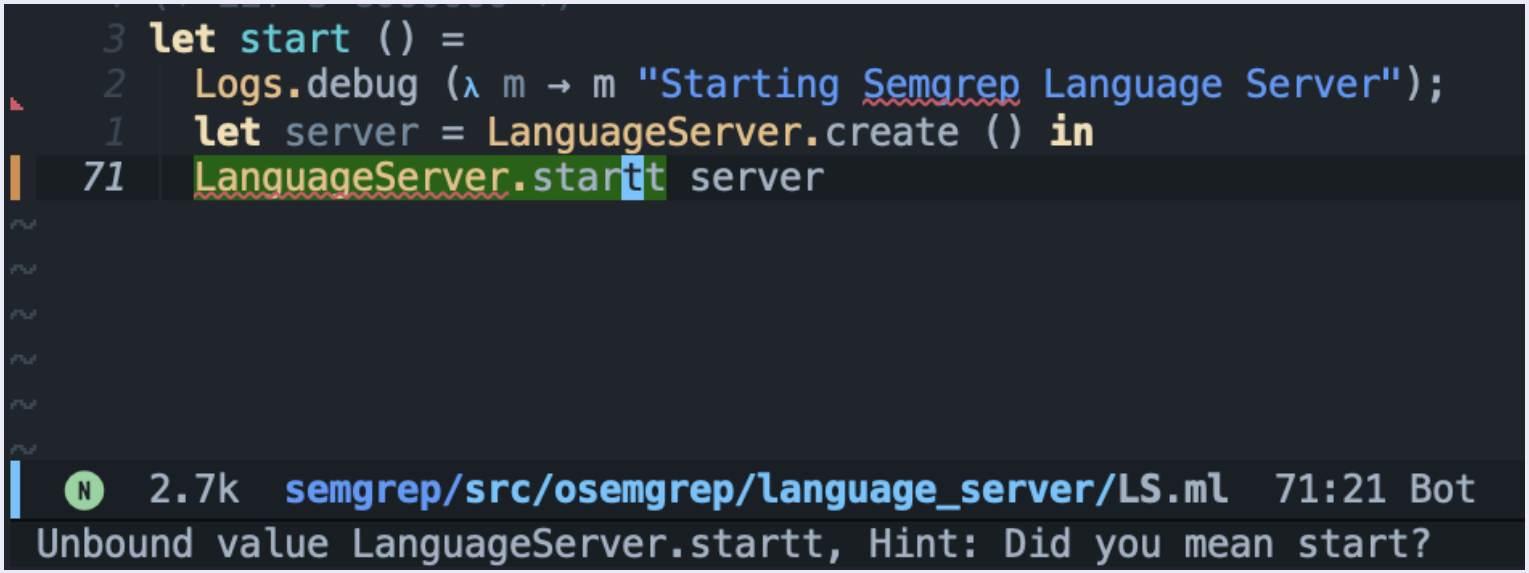
How do we show security bugs early?

In the editor!



How do we show security bugs early?

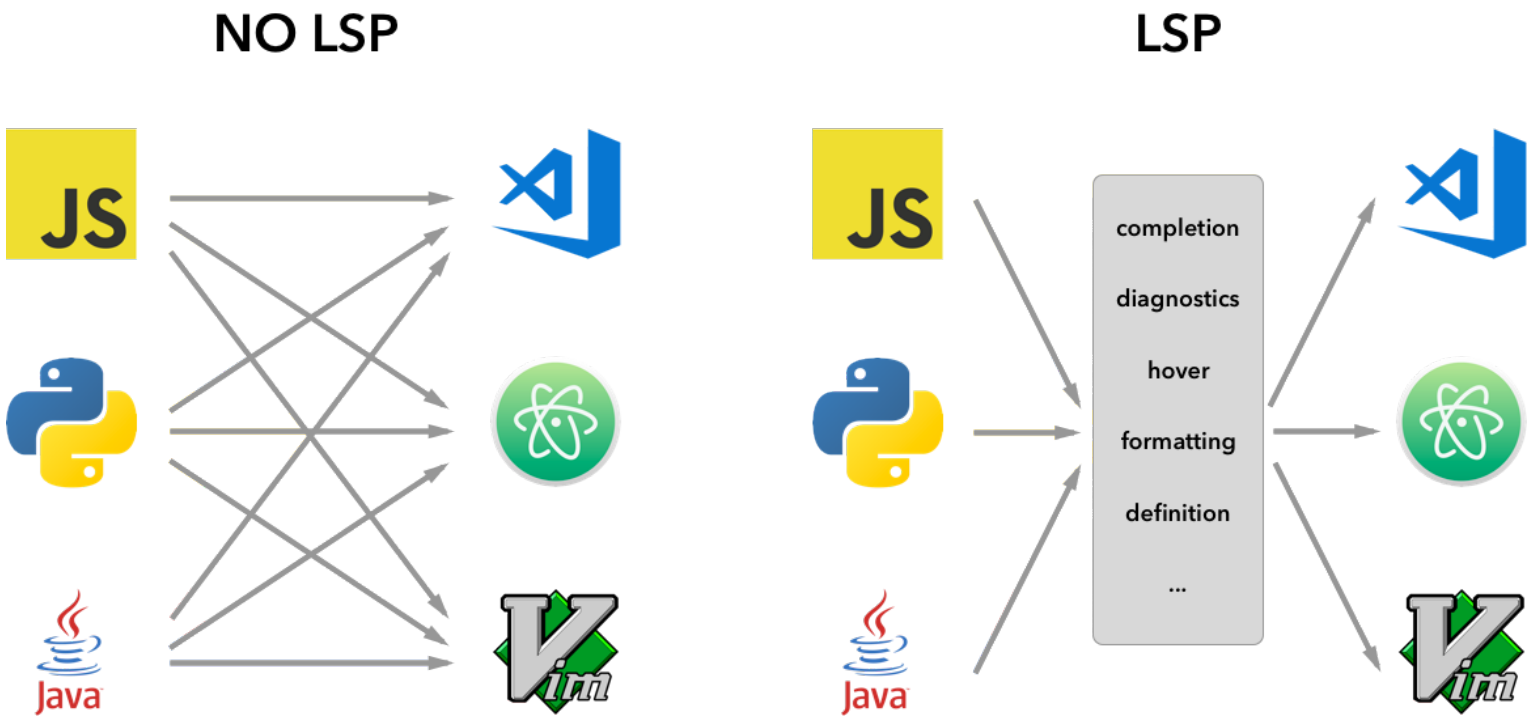
In the editor!



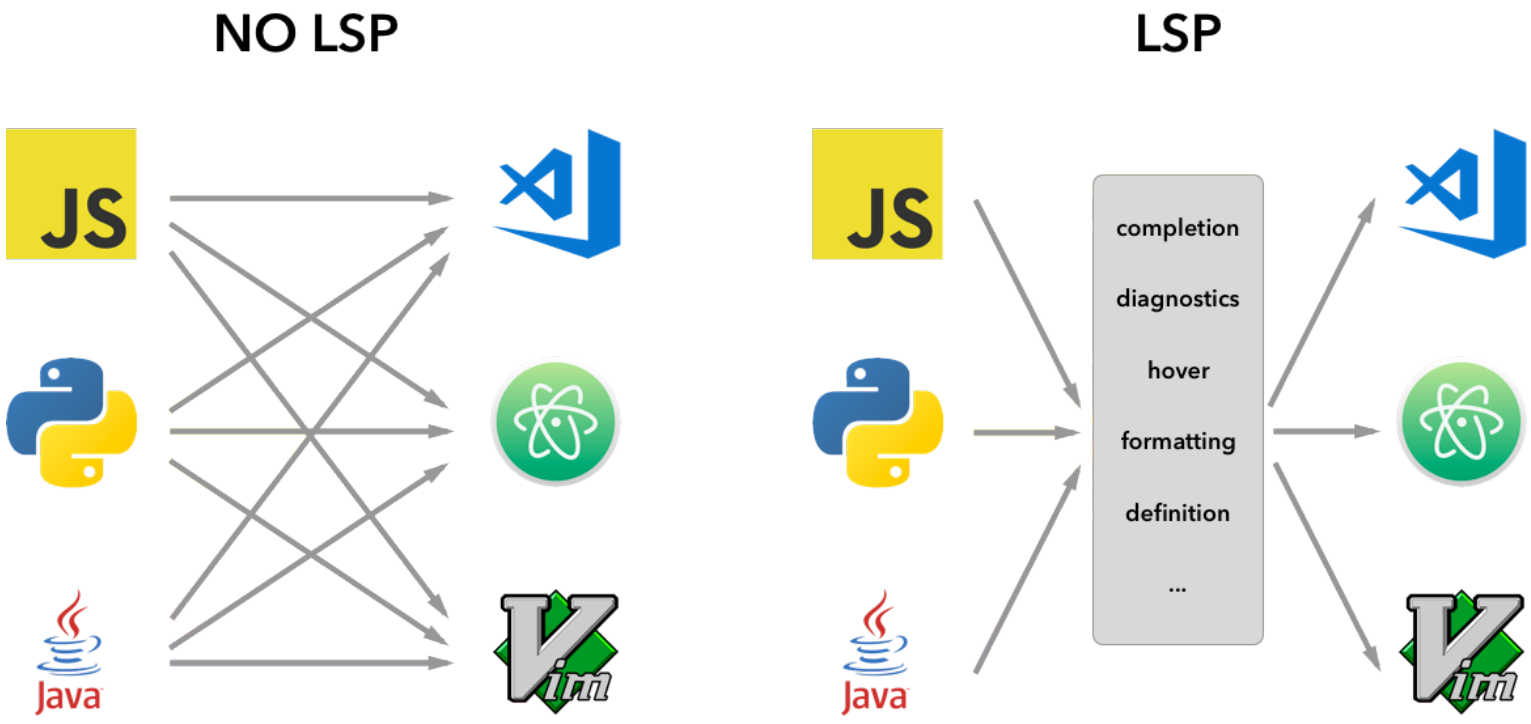
Goals

- Provide a similar user experience to normal language checking
- Abstract away editing and language features for editors to one code base

What is the Language Server Protocol

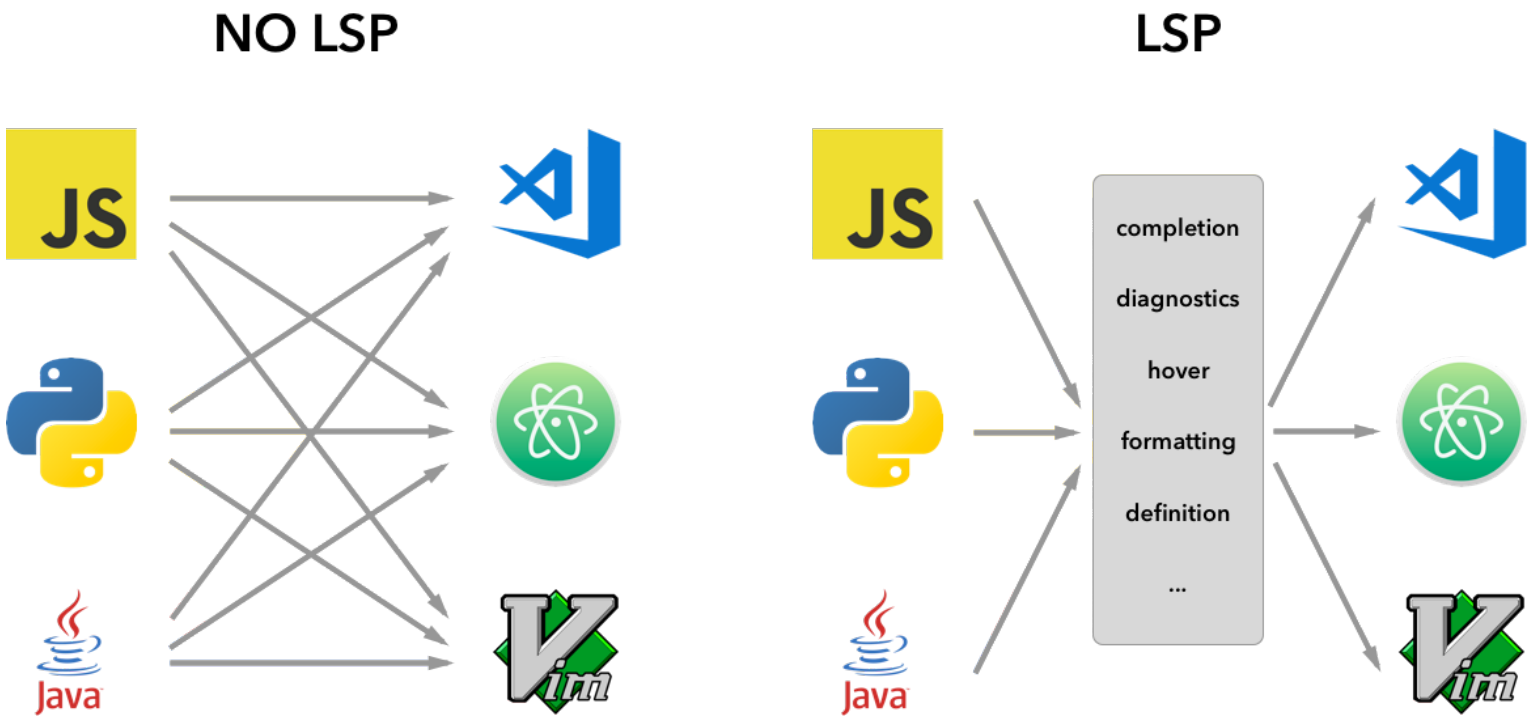


What is the Language Server Protocol



Language Server Protocol (LSP) Defines all the ways an editor can interact with a LS

What is the Language Server Protocol



Language Server Protocol (LSP) Defines all the ways an editor can interact with a LS

Language Server (LS) Program that provides language tooling (syntax errors, completion, refactoring etc.)

Language Client The development tool/editor etc. handles documents, user interaction etc. (i.e. Emacs)

Case study: Rust Analyzer

- Rust Analyzer: A language server for the Rust language
- Rust takes a long time to compile
- Rust Analyzer provides feedback instantly

Case study: Rust Analyzer

- Rust Analyzer: A language server for the Rust language
- Rust takes a long time to compile
- Rust Analyzer provides feedback instantly

Provides the editor

- Compiler errors/warnings
- Potential fixes
- Completion
- Type signatures
- Auto imports
- View dependency graph
- Run code/tests
- Refactoring
- Much much more

Case study: Rust Analyzer

- Rust Analyzer: A language server for the Rust language
- Rust takes a long time to compile
- Rust Analyzer provides feedback instantly

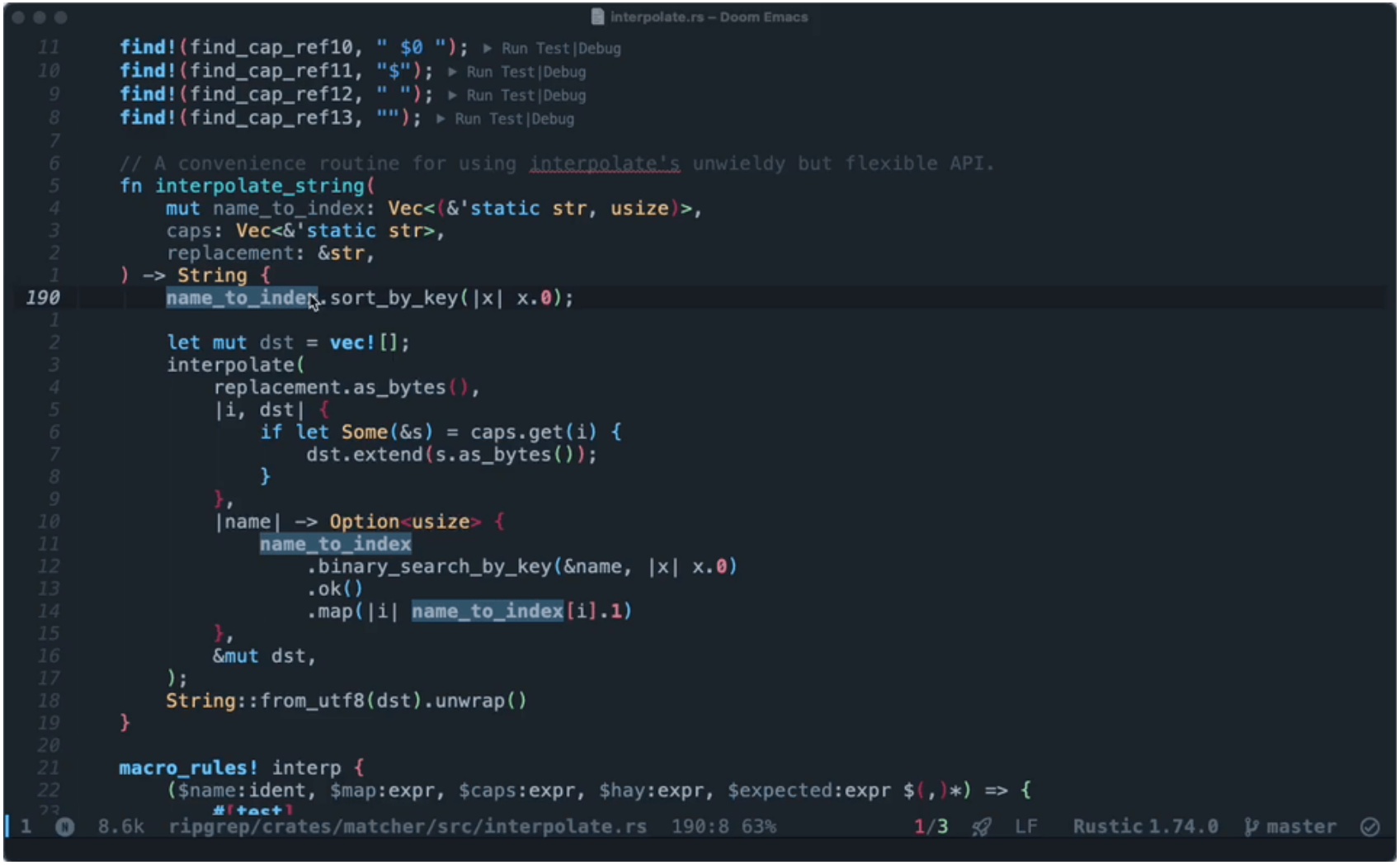
Provides the editor

- Compiler errors/warnings
- Potential fixes
- Completion
- Type signatures
- Auto imports
- View dependency graph
- Run code/tests
- Refactoring
- Much much more

TL;DR

Developing Rust with Rust Analyzer is a pleasure, and makes dealing with advanced language features significantly easier

Rust Analyzer in action



Why is this useful?

User perspective

- Same experience across editors
- Can easily setup and use LS' made for other editors, if developers don't support a certain editor
- Performance is not dependent on editor
- Bug fixes, updates, etc. all come out at the same time

Why is this useful?

User perspective

- Same experience across editors
- Can easily setup and use LS' made for other editors, if developers don't support a certain editor
- Performance is not dependent on editor
- Bug fixes, updates, etc. all come out at the same time

Developer perspective

- Adding new editors is quick and easy
- Only need one mental model
- Write tests for the LS, not for the editor

So what about Emacs?

- Gets to benefit from work put into other editors

So what about Emacs?

- Gets to benefit from work put into other editors
- Language tooling, the CPU intensive part of editors, can be written in something else
  - Lisp is fast, but not that fast

So what about Emacs?

- Gets to benefit from work put into other editors
- Language tooling, the CPU intensive part of editors, can be written in something else
  - Lisp is fast, but not that fast
- `lsp-mode`, an LSP client, is commonly included in popular Emacs distributions (Spacemacs, Doom Emacs, etc.)

So what about Emacs?

- Gets to benefit from work put into other editors
- Language tooling, the CPU intensive part of editors, can be written in something else
  - Lisp is fast, but not that fast
- `lsp-mode`, an LSP client, is commonly included in popular Emacs distributions (Spacemacs, Doom Emacs, etc.)
- Emacs 29 includes `eglot-mode` built-in, a LSP client
  - Lighter weight than `lsp-mode`

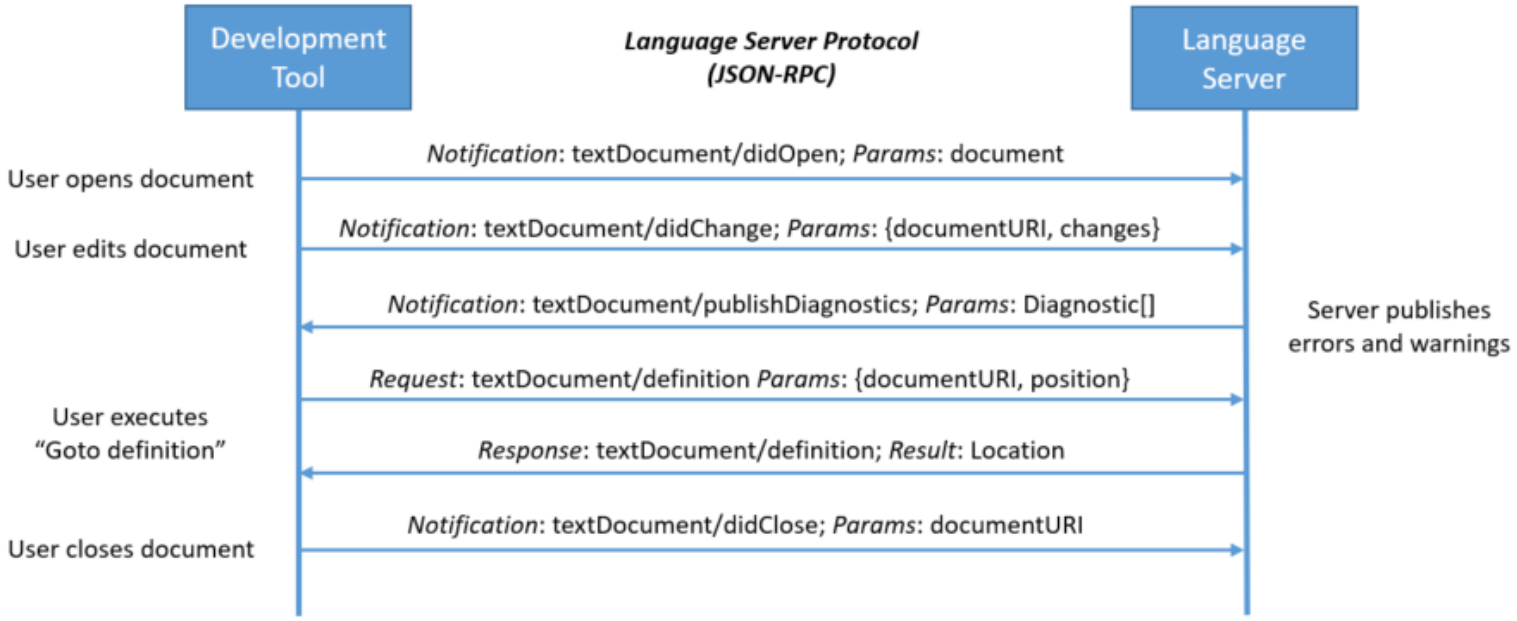
So what about Emacs?

- Gets to benefit from work put into other editors
- Language tooling, the CPU intensive part of editors, can be written in something else
  - Lisp is fast, but not that fast
- `lsp-mode`, an LSP client, is commonly included in popular Emacs distributions (Spacemacs, Doom Emacs, etc.)
- Emacs 29 includes `eglot-mode` built-in, a LSP client
  - Lighter weight than `lsp-mode`

Some supported languages

C/C++/C#, Python, Rust, Type/Javascript, Dockerfile, Elixir, D, Java, Haskell, Ruff, Semgrep, and more

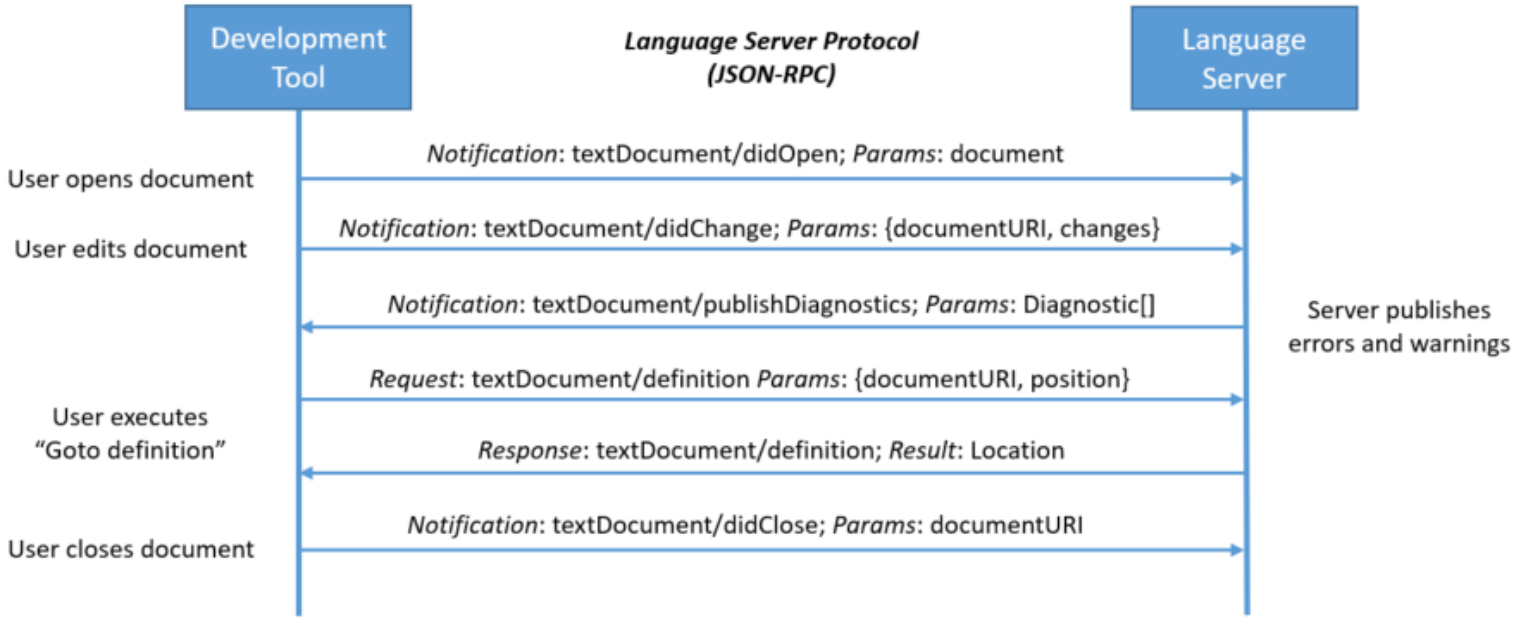
Brief communication overview



JSONRPC A JSON based remote procedure call protocol (think http but JSON)

- A way for two programs to communicate
- Transport platform agnostic (can be stdin/out, sockets etc.)

Brief communication overview

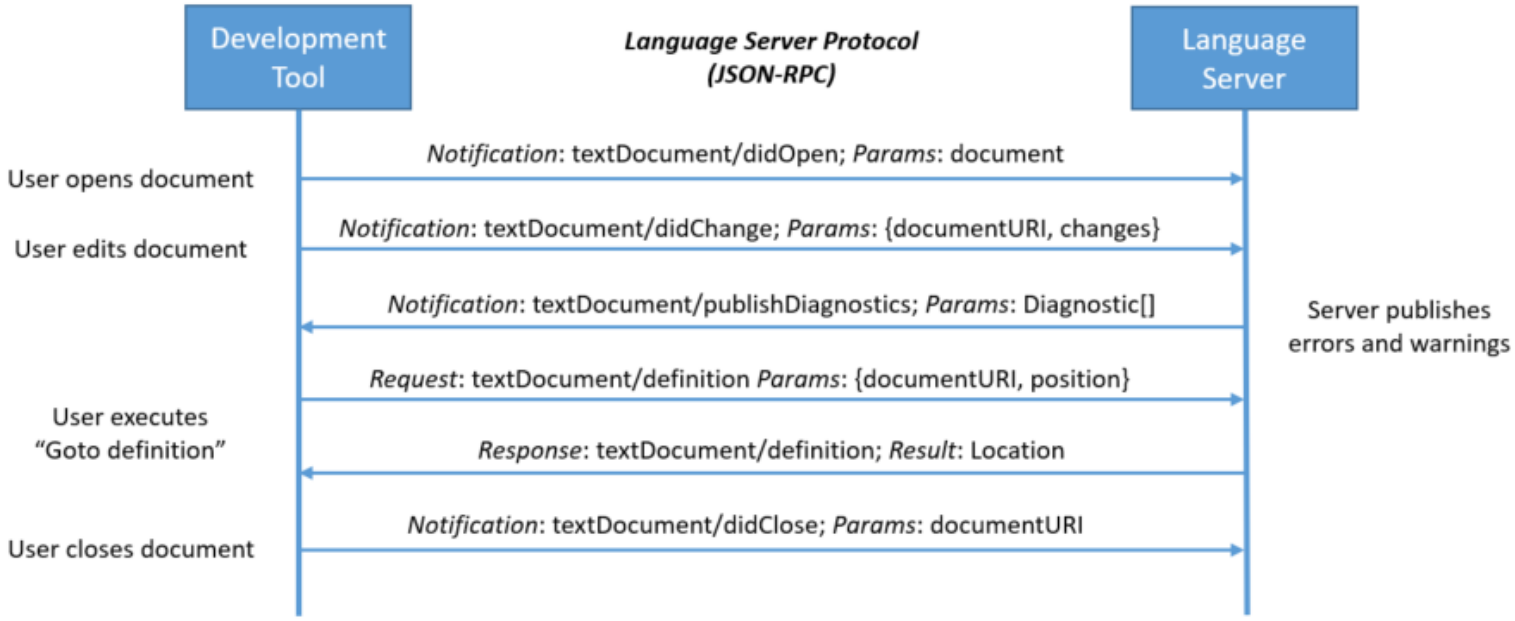


JSONRPC A JSON based remote procedure call protocol (think http but JSON)

- A way for two programs to communicate
- Transport platform agnostic (can be stdin/out, sockets etc.)

Request message that requires a response from the other party

Brief communication overview



JSONRPC A JSON based remote procedure call protocol (think http but JSON)

- A way for two programs to communicate
- Transport platform agnostic (can be stdin/out, sockets etc.)

Request message that requires a response from the other party

Notification message that does not expect a response

Example Request

```
{
    "jsonrpc": "2.0",
    "id" : 1,
    "method": "textDocument/definition",
    "params": {
        "textDocument": {
            "uri": "file:///p%3A/mseng/VSCode/Playgrounds/cpp/use.cpp"
        },
        "position": {
            "line": 3,
            "character": 12
        }
    }
}
```

Example Response


```
{
    "jsonrpc": "2.0",
    "id": 1,
    "result": {
        "uri": "file:///p%3A/mseng/VSCode/Playgrounds/cpp/provide.cpp",
        "range": {
            "start": {
                "line": 0,
                "character": 4
            },
            "end": {
                "line": 0,
                "character": 11
            }
        }
    }
}
```

LSP Capabilities

- Almost all of the LSP is opt-in
- Server and client communicate on what parts of the protocol they both support
- Custom capabilities are possible too - just define a custom JSONRPC method

LSP Capabilities

- Almost all of the LSP is opt-in
- Server and client communicate on what parts of the protocol they both support
- Custom capabilities are possible too - just define a custom JSONRPC method

Some relevant capabilities

- Open/Close file
- Diagnostics
- Code Actions
- Completion
- Other actions
  - Hover
  - Signature
  - Symbols

LSP Capabilities

- Almost all of the LSP is opt-in
- Server and client communicate on what parts of the protocol they both support
- Custom capabilities are possible too - just define a custom JSONRPC method

Some relevant capabilities

- Open/Close file
- Diagnostics
- Code Actions
- Completion
- Other actions
  - Hover
  - Signature
  - Symbols

Custom Capabilities

- Example: Rust Analyzer has "Structural Search and Replace" request
- If you choose to go down this route, you must implement this custom capability in every client

Tips on writing a LS

- I wrote Semgrep's in OCaml since our codebase was almost all OCaml already

Tips on writing a LS

- I wrote Semgrep's in OCaml since our codebase was almost all OCaml already
- Structure is similar to a REST server (i.e. a bunch of independent endpoints)

Tips on writing a LS

- I wrote Semgrep's in OCaml since our codebase was almost all OCaml already
- Structure is similar to a REST server (i.e. a bunch of independent endpoints)
- Would recommend Typescript or Rust depending on level of performance desired
  - Typescript has a lot of support, and documentation
  - Rust is fast, but is going to take a lot more effort

Tips on writing a LS

- I wrote Semgrep's in OCaml since our codebase was almost all OCaml already
- Structure is similar to a REST server (i.e. a bunch of independent endpoints)
- Would recommend Typescript or Rust depending on level of performance desired
  - Typescript has a lot of support, and documentation
  - Rust is fast, but is going to take a lot more effort
- The hard part is not LSP, but the actual logic

Tips on writing a LS

- I wrote Semgrep's in OCaml since our codebase was almost all OCaml already
- Structure is similar to a REST server (i.e. a bunch of independent endpoints)
- Would recommend Typescript or Rust depending on level of performance desired
  - Typescript has a lot of support, and documentation
  - Rust is fast, but is going to take a lot more effort
- The hard part is not LSP, but the actual logic
- If you want to do analysis on code, you'll need some sort of parser, a way to get errors etc.

Tips on writing a LS

- I wrote Semgrep's in OCaml since our codebase was almost all OCaml already
- Structure is similar to a REST server (i.e. a bunch of independent endpoints)
- Would recommend Typescript or Rust depending on level of performance desired
  - Typescript has a lot of support, and documentation
  - Rust is fast, but is going to take a lot more effort
- The hard part is not LSP, but the actual logic
- If you want to do analysis on code, you'll need some sort of parser, a way to get errors etc.
- If you're adapting an existing language tool, this is much easier

Supporting a LS through LSP mode in Emacs

- How to write a client for a Language Server in Emacs
- There's the official `lsp-mode` repository on Github w/ many clients
  - These are available to anyone who installs `lsp-mode`
  - Alternatively, one can create a separate package altogether (but we won't focus on this)
- Steps:
  1. A simple .el file that contains the logic
  2. An entry into the list of clients
  3. Documentation!

Create a client

```
;; lsp-mode/clients/lsp-<client-name>.el
;;
;; Use lsp-mode library
(require 'lsp-mode)

(lsp-register-client
 (make-lsp-client
  :new-connection (lsp-stdio-connection '("<server-executable>"))
  :activation-fn (lsp-activate-on "<language-name>")
  :server-id 'language-server-name))

(lsp-consistency-check 'language-server-name)
(provide 'lsp-client-name)
```

Add to list of client packages

```
;; lsp-mode/lsp-mode.el
;; a bunch of lisp
(defcustom lsp-client-packages
  '(
    ;; A bunch of clients
    lsp-client-name
    ;; more clients
    )
  "List of the clients to be automatically required."
  :group 'lsp-mode
  :type '(repeat symbol))
;; more lisp
```

Add to list of clients

```
// lsp-mode/docs/lsp-clients.json
{
  // other clients...
  "name": "client name",
  "full-name": "full client name",
  "server-name": "language-server-name",
  "server-url": "<url>",
  "installation": "<installation command>",
  "debugger": "Yes or Not available"
}
```

Add documentation!

```
# lsp-mode/mkdocs.yml
# Documentation!
# Other client pages
- Client Name: page/lsp-<client-name>.md
# More client pages
```

Adding commands and custom capabilities

```
;; Custom notification
(defun client-notify-command (params)
 "Documentation"
 (interactive)
 (lsp-notify "method" params))

;; Custom request
(defun client-request-command (params)
 "Documentation"
 (interactive)
 (lsp-request-async "method" params
                    (lambda (result)
                      (do-thing result))))
;; Previous content of lsp-client-name.el
```

Thanks for listening

Resources

- Semgrep: We're hiring!
- LSP Specification
- lsp-mode + docs
- Rust Analyzer
- Long Video Tutorial

Q&A Time