A Lightweight Framework Approach to Building Programming Language Editor Support

Alexandre Lachance[1,2], Sébastien Mosser PhD[1,2]

[1]Computing and Software, McMaster University, Hamilton, Canada. [2]Centre for Software Certification, McMaster University, Hamilton, Canada.

Introduction

- Programming languages used by smaller communities, such as very specialized programming languages, often don't have the same level of **editor support** (i.e. auto-completion, syntax highlighting) as more popular languages do.
- Current "solutions" all require the entire language to be built with them to get editor support as an artifact. They also have no way to interact with **existing tools**.
- The modern approach to editor support is the usage of the **Language Server Protocol (LSP)**. Language servers allow for editor-agnostic language support. They provide features like auto-complete, go to definition, or renaming [1]. The problem being the need for a high-level of programming language expertise for their development.

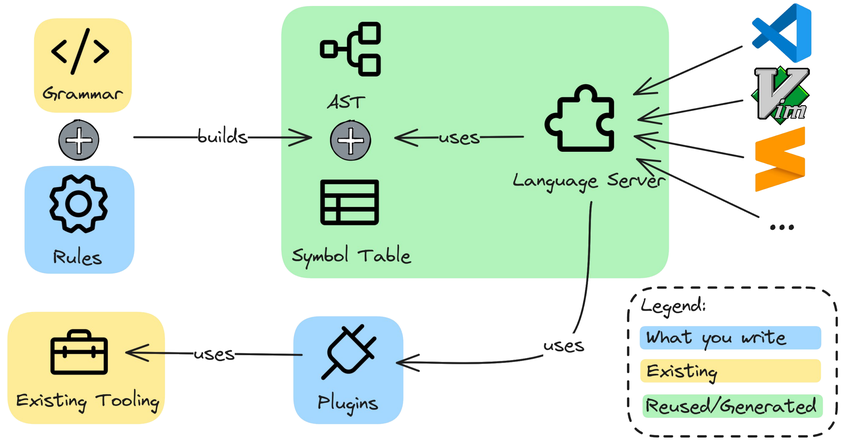
Objectives

- Provide a **lightweight framework** to enable language creators to focus on their language without the burden of developing a complete IDE.
- Provide a way for the new editor tooling to interact with **existing language tools** (i.e. a compiler or a static analyzer).

Current "Solutions"

langium, MetaCase, Xtext, MPS

Design of the framework



- Language servers usually use two main data structures as a basis for most of their services: an **Abstract Syntax Tree (AST)** and a **Symbol Table**. Our approach consists of reusing these data structures and for the language server to solely rely on them.
- To generate these structures the framework relies on 2 main inputs: an existing **grammar** and a set of **rules**.
- The **rules** are written using an off-the-shelf configuration language. They act as an annotation over the Concrete Syntax Tree (CST) parsed with the **grammar**. They provide the missing language specific information necessary to build an **AST** and a **symbol table** for each source code file.
- We also provide a plugin interface. **Plugins** act as adapters from **existing language tools** to language server compatible data structures (e.g. getting compiler errors in your IDE).

```
Rule(
  node_name: "ConstantDeclaration",
  symbol: Init(type: "Constant", name_node: "Name", type_node: "Type"),
  children: [
    (query: Field("type"), rule: Rule("Type")),
    (query: Field("name"), rule: Direct("Name")),
    (query: Field("value"), rule: Rule("Expression")),
  ]
)
```

```
const bit<16> TYPE_IPV4 = 0x800;
```

Key facts

- **Purpose:** Provide a simple way to build powerful editor support for the target programming language.
- **Features:** Out of the box support for Auto-completion, Syntax Highlighting, Go To Definition, Renaming, and much more.
- **Compatibility:** Integrates with existing tooling via a custom plugin system.
- **Configuration:** Rules are written in an off-the-shelf JSON-like configuration language, avoiding the need for a new proprietary language.

Results

- The framework has been validated on 3 different languages:
  - **P4:** "Programming Protocol-independent Packet Processors (P4) is a domain-specific language for network devices, specifying how data plane devices (switches, NICs, routers, filters, etc.) process packets." [2]
  - **JPipe:** A in-house research language focusing on the justification of pipelines. Compiles to diagrams.
  - **Protobuf: "**Protocol Buffers are language-neutral, platform-neutral extensible mechanisms for serializing structured data." [3]
- Dissemination:
  - **MDENet** presentation [4]
  - **P4.org Open Source Developer Days** presentation
  - Used at **Kaloom™ Networks** (a Montreal startup)
  - Presentation at the 2023 **P4 Workshop**

Conclusion

- By abstracting away the common and complex parts of language servers, we built a **lightweight framework** for building programming language editor support.
- This approach has been validated with the successful implementation of the framework targeting **3 different programming languages**.
- Next steps:
  - Improve coverage of Language Server Protocol (LSP) features.
  - Use **generative programming** to improve language server performance.

Validation

P4 (Linux Foundation) **jPipe** (McSCert) **protobuf** (Google)

References

[1] https://microsoft.github.io/language-server-protocol
[2] https://p4.org/
[3] https://protobuf.dev/
[4] https://youtu.be/JzCYxz4G_Cc

Alexandre Lachance
McMaster University, Computing & Software
Email: lachaa2@mcmaster.ca

BRIGHTER WORLD