Title

Federico Cristiano Bruzzone

Id. Number: 27427A

MSc in Computer Science

Advisor:	Prof. Walter Cazzola
Co-Advisor:	Dr. Luca Favalli

UNIVERSITÀ DEGLI STUDI DI MILANO

Computer Science Department

ADAPT-Lab

Academic Year 2023-2024

COMPUTER SCIENCE DEPARTMENT • ADAPT-LAB

Contents

1 Introduction

2 Background

In this chapter, we provide an overview of the concepts and technologies that are relevant to the work presented in this thesis. We start by introducing the concept of language servers and the Language Server Protocol (LSP) in Section 2.1. We then discuss language workbenches in Section 2.2, type systems in Section 2.3, and software and language product lines in Section 2.4.

The goal of this chapter is to provide the reader with the necessary background knowledge to understand the work presented in the following chapters. We assume that the reader has a basic understanding of programming languages and software development.

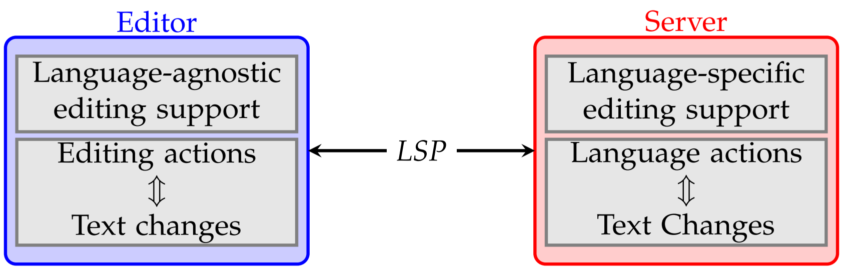
2.1 Language Server Protocol

The Language Server Protocol[1] (LSP) is a protocol that allows for the communication between a language server and an editor or an IDE. The LSP is used to decuple the a language-agnotic editor or integrated development environment (IDE) from the language-specific features of a language server (see Listing 2.1). This allows for the development of language servers that can be used with multiple editors or IDEs. The LSP is based on the stateless JSON-RPC protocol and defines a set of messages that are used to communicate between the language server and the editor or IDE.

Usually, the LSP Clients are developed as plugins for popular editors or IDEs decresing the effort to support a new language in a given editor. The LSP Clients are responsible for sending requests to the language server and processing the responses. The language server is responsible for providing language-specific features. The LSP defines a set of messages that are used to communicate between the language server and the editor or IDE. These messages include requests for code completion, code navigation, and code analysis, as well as notifications for changes to the document, diagnostics, and progress reports.

Language servers are *de facto* standard for providing language-specific features in editors and IDEs. The LSP is supported by a wide range of editors and IDEs, including Noevim[2], Visual Studio Code[3], Eclipse[4], and IntelliJ IDEA[5]. There are several language

[1] `https://microsoft.github.io/language-server-protocol`
[2] `https://neovim.io/doc/user/lsp.html`
[3] `https://code.visualstudio.com/api/language-extensions/language-server-extension-guide`
[4] `https://www.eclipse.org/community/eclipse_newsletter/2017/may/article1.php`
[5] `https://plugins.jetbrains.com/docs/intellij/language-server-protocol.html`



Listing 2.1. *LSP approach to language support. Borrowed from [2].*

servers available for popular programming languages, including Rust, TypeScript, Python, and Java and most of them are open-source[6].

The LSP is initiated by Microsoft and is now an open standard that is maintained by the Language Server Protocol Working Group. It was designed for the use with the Visual Studio Code editor, but it has since been adopted by other editors and IDEs. The LSP is under open-source license and is available on GitHub[7].

2.1.1 JSON-RPC

The LSP uses JSON-RPC to communicate between a language server and an editor. JSON-RPC (v2)[8] is a stateless, light-weight remote procedure call (RPC) [1] protocol that uses JSON as the data format.

RPC is a protocol that allows a client to call a procedure on a remote server. The client sends a request to the server, and the server sends a response back to the client. The JSON-RPC protocol defines a set of messages that are used to communicate between the client and the server. These messages include requests, responses, and notifications. The JSON-RPC protocol is designed to be simple and easy to implement, making it well-suited for use in web applications and other distributed systems.

JSON-RPC is a JSON based implementation of the RPC protocol. It defines a set of rules for encoding and decoding JSON data, as well as a set of rules for *Request*, *Notification*, and *Response* messages. The messages are sent over a transport layer, such as HTTP or WebSockets. The JSON-RPC protocol is designed to be simple and easy to implement, making it well-suited for use in web applications and other distributed systems.

All messages refer to a *method* that is a string containing the name of the method to be called. The *params* field is an array or object containing the parameters to be passed to the method. Typically, messages are synchronous, meaning that the client waits for a response from the server before continuing. The *id* field is a unique identifier for the message, which is used to match requests with responses. However, the JSON-RPC protocol also supports asynchronous messages, known as notifications, which do not

[6]`https://microsoft.github.io/language-server-protocol/implementors/servers`
[7]`https://github.com/microsoft/language-server-protocol`
[8]`https://www.jsonrpc.org/specification`

require a response from the server. This is implemented by setting the *id* field to *null*, in which case the server does not send a response back to the client.

The JSON-RPC specification includes the ability for clients to batch multiple requests or notifications by sending them as a list. The server is expected to respond with a corresponding list of results for each request. Additionally, the server has the flexibility to process these requests concurrently.

2.2 Language Workbenches

2.3 Type Systems

2.4 Software and Language Product Lines

3 Concept

4 Implementation

5 Evaluation

6 Related Work

7
Conclusions

Bibliography

[1] Andrew D. Birrell and Bruce Jay Nelson. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems*, 2(1):39–59, February 1984.

[2] Roberto Rodriguez-Echeverria, Javier Luis Cánovas Izquierdo, Manuel Wimmer, and Jordi Cabot. Towards a language server protocol infrastructure for graphical modeling. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, MODELS '18, page 370–380, New York, NY, USA, 2018. Association for Computing Machinery.