Toward a Modular Approach for Type Systems and LSP Generation

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

1 Institution where the internship work was carried out

My internship was carried out at the **ADAPT-Lab** of the Università degli Studi di Milano, under the supervision of Prof. Walter **Cazzola** and Dr. Luca **Favalli**. The lab is part of the Department of Computer Science and is focused on research in the field of programming languages and software engineering.

2 Initial context

The landscape of software development has been undergoing rapid transformation, driven by the increasing complexity and diversity of programming languages and tools. Among the key advancements, the **Language Server Protocol** (LSP) has emerged as a crucial technology, fundamentally reshaping how developers interact with their development environments. Before the advent of LSP, each Integrated Development Environment (IDE) had to implement its own set of language-specific features such as auto-completion, error checking, and refactoring. This approach often resulted in a *fragmented* and *inconsistent* development experience, requiring substantial effort to maintain and update these features across different environments. **Language Workbenches** (LWs), thanks to their modular and extensible nature, could have been used to streamline the development of type systems and language-specific features.

3 Work Objectives

This section outlines the primary goals of the study, focusing on foundational concepts and technologies behind **Language Server Protocol** LSP. Emphasis is placed on **static analysis**, **type systems** (TSs), and the integration of **compilers** and **language workbenches** (LWs), along with the benefits and challenges of **domain-specific languages** (DSLs) and **modular architectures**.

1. *Examine the foundational concepts and technologies underlying **LSP***: This includes understanding the components of LSP, such as JSON-RPC and command specifications.
2. *Explore static analysis and **TSs***: Delve into the theoretical and practical aspects of these technologies and their integration into language servers to enhance code quality and developer productivity.
3. *Investigate the role of **compilers** and **LWs** in LSP development*: This involves exploring how compiler phases and language workbenches can be leveraged to integrate LSP.
4. *Analyze the benefits and challenges of **DSLs***: This includes examining the advantages of DSLs in the context of LSP, as well as the challenges associated with designing and implementing DSLs for the **TSs** domains.
5. *Assess the impact of modular architectures in **TSs** and **LSP** development*: Evaluate how modularity, as promoted by LSP and related tools, supports the efficient

development and maintenance of programming languages and their features.

6. *Reducing the number of combinations to support* **L** *languages*: Investigate how modularity could reduce the number of combinations, moving from $\mathbf{L} + \mathbf{E}$ to $\mathbf{L} \times 1$ to streamline LSP development.
7. *Demonstrate practical implementations and* ***case studies***: Provide detailed examples and case studies to illustrate the real-world applications and benefits of the proposed concepts and technologies.

4 Description of work performed

The work undertaken in this study involved a comprehensive exploration and practical implementation of the foundational concepts and technologies behind the **Language Server Protocol** (LSP). The primary focus was to generate the integration and functionality of LSP for every possible language written in Neverlang, to create a more cohesive and efficient development experience.

Initially, an *in-depth* analysis of LSP components was conducted, emphasizing the role of JSON-RPC as a communication protocol and the various command specifications that define the interactions between IDEs and language servers. This foundational understanding was crucial for developing robust solutions that could seamlessly integrate into diverse development environments.

The research also delved into **static analysis** and **type systems**, exploring both theoretical frameworks and practical applications. These investigations were critical for incorporating advanced code quality checks and developer productivity enhancements into the language servers. By integrating these elements, the project aimed to provide real-time feedback and error detection, significantly improving the coding experience.

A significant portion of the work focused on the interplay between **compilers** and **language workbencheis** (LWs) in the context of LSP development. The study examined how different phases of compiler operations could be leveraged to support LSP functionalities, ensuring that language-specific features were accurately implemented and maintained. Language workbenches, with their modular and extensible architectures, were utilized to streamline the creation and customization of these features.

The research further investigated the benefits and challenges associated with **domain-specific languages** (DSLs). By designing and implementing DSLs tailored to specific type system domains, the study aimed to demonstrate how these specialized languages could enhance developer productivity and code quality. The challenges of DSL design and implementation were also addressed, providing insights into effective strategies for overcoming these obstacles.

Modularity was a key theme throughout the project, with a strong emphasis on how **modular architectures** could support the efficient development and maintenance of programming languages and their features. This modular approach not only facilitated code reusability and scalability but also played a crucial role in reducing the number of combinations needed to support multiple languages, thereby streamlining LSP development. In fact, we reduce the number of combinations ($\mathbf{L} + \mathbf{E}$) required to support LSP for **L** languages and **E** editors to $\mathbf{L} \times 1$ combinations, generating the **Clients**.

To illustrate the practical applications and benefits of the proposed concepts and technologies, detailed case studies and practical implementations were provided (e.g., LSP and Type System integration for Neverlang and Simplelang). These examples showcased the real-world impact of the research and highlighted the potential for enhancing software development workflows through advanced language server capabilities.

5 Technologies involved

The work involved the following technologies:

- **Language Server Protocol (LSP)**: A standardized communication protocol that enables the integration of language-specific features into IDEs.
- **JSON-RPC**: A lightweight remote procedure call protocol that serves as the foundation for LSP communication.
- **Compilers and Language Workbenches (LWs)**: Tools and frameworks used to analyze, transform, and generate code, as well as to develop domain-specific languages and language features.
- **Type Systems**: Theoretical frameworks and practical implementations that govern data types and their interactions in programming languages.
- **Domain-Specific Languages (DSLs)**: Specialized languages designed to address specific problem domains and enhance developer productivity.
- **LSP4J**: A Java library that provides support for implementing language servers using the LSP.
- **Software Product Lines (SPLs)**: A methodology for managing feature variability across software product variants.
- **Programming Languages**: Including Java, Neverlang, TypeScript, JavaScript, and other languages.

6 Competences and achievements

We are writing a paper for *Journal of Systems and Software* called
"**Code Less to Code More**: Streamlining LSP Development for Language Families".

During the course of this research, significant competences and achievements have been attained. The key competences and achievements include:

- *Library Development*: A comprehensive software library was designed and implemented, leveraging Java programming language to encapsulate essential functionalities required for effective LSP integration. This library encompasses modules for command specifications, JSON-RPC communication, key method implementations, and advanced source code analysis capabilities for type systems. The library comprises approximately 10000 lines of code.

- *LSP Enhancement*: By focusing on extending LSP capabilities, the library enables seamless communication between IDEs and language servers, fostering improved code navigation, syntax highlighting, code completion, and real-time error detection. These enhancements significantly elevate the development experience by providing developers with powerful tools to streamline software development workflows.
- *Modular Design*: The library was architected with a modular design approach, facilitating easy integration and scalability across different IDE environments and programming languages. This modular architecture promotes code reusability, maintenance efficiency, and adaptability to evolving software requirements.
- *Type System Integration*: Integral to the library's functionality is its robust type system integration, which enhances program analysis and verification capabilities. The implementation includes sophisticated mechanisms for type inference, symbol table management, and scope resolution, ensuring accurate and efficient handling of language-specific constructs.
- *A new DSL for Type Systems and LSP implementation*: A new DSL was designed and implemented to support the definition of type systems and LSP features. This DSL provides a high-level abstraction for specifying language-specific constructs and interactions, enabling rapid development and customization of type systems and language features.
- *Contributions to Language Engineering*: Through the development of this library, significant contributions have been made to the field of language engineering. By advancing LSP capabilities and promoting best practices in software language design and implementation, the research underscores its commitment to improving software development practices and fostering innovation in language tooling.

Bibliography

[1] Djonathan Barros, Sven Peldszus, Wesley K. G. Assunção, and Thorsten Berger. Editing Support for Software Languages: Implementation Practices in Language Server Protocols. In Manuel Wimmer, editor, *Proceedings of the 25th International Conference on Model Driven Engineering Langauges and Systems (MoDELS'22)*, pages 232–243, Montréal, Canada, October 2022. ACM.

[2] Francesco Bertolotti, Walter Cazzola, and Luca Favalli. SPꓶꓶ98: Software Product Lines Extraction Driven by Language Server Protocol. *Journal of Systems and Software*, 205, November 2023.

[3] Lorenzo Bettini. *Implementing Domain-Specific Languages with Xtext and Xtend*. PACKT Publishing Ltd, August 2013.

[4] Hendrik Bünder. Decoupling language and editor-the impact of the language server protocol on textual domain-specific languages. 2019.

[5] Walter Cazzola and Luca Favalli. Towards a Recipe for Language Decomposition: Quality Assessment of Language Product Lines. *Empirical Software Engineering*, 27(4), April 2022.

[6] Walter Cazzola and Diego Mathias Olivares. Gradually Learning Programming Supported by a Growable Programming Language. *IEEE Transactions on Emerging Topics in Computing*, 4(3):404–415, September 2016. Special Issue on Emerging Trends in Education.

[7] Luca Favalli, Thomas Kühn, and Walter Cazzola. Neverlang and FeatureIDE Just Married: Integrated Language Product Line Development Environment. In Philippe Collet and Sarah Nadi, editors, *Proceedings of the 24th International Software Product Line Conference (SPLC'20)*, pages 285–295, Montréal, Canada, 19th-23rd of October 2020. ACM.

[8] Martin Fowler and Rebecca Parsons. *Domain Specific Languages*. Addison Wesley, September 2010.

[9] Øystein Haugen, Birger Møller-Pedersen, Jon Oldevik, Gøran K. Olsen, and Andreas Svendsen. Adding Standardized Variability to Domain Specific Languages. In Klaus Pohl and Birgit Geppert, editors, *Proceedings of the 12th International Software Product Line Conference (SPLC'08)*, pages 139–148, Limerick, Ireland, September 2008. IEEE.

[10] Thomas Kühn and Walter Cazzola. Apples and Oranges: Comparing Top-Down and Bottom-Up Language Product Lines. In Rick Rabiser and Bing Xie, editors, *Proceedings of the 20th International Software Product Line Conference (SPLC'16)*, pages 50–59, Beijing, China, 19th-23rd of September 2016. ACM.

[11] Thomas Kühn, Walter Cazzola, and Diego Mathias Olivares. Choosy and Picky: Configuration of Language Product Lines. In Goetz Botterweck and Jules White, editors, *Proceedings of the 19th International Software Product Line Conference (SPLC'15)*, pages 71–80, Nashville, TN, USA, 20th-24th of July 2015. ACM.

[12] Thomas Kühn, Walter Cazzola, Nicola Pirritano Giampietro, and Massimiliano Poggi. Piggyback IDE Support for Language Product Lines. In Thomas Thüm and Laurence Duchien, editors, *Proceedings of the 23rd International Software Product Line Conference (SPLC'19)*, pages 131–142, Paris, France, 9th-13th of September 2019. ACM.

[13] Manuel Leduc, Thomas Degueule, Eric Van Wyk, and Benoît Combemale. The Software Language Extension Problem. *Software and Systems Modeling*, 19(2):263–267, January 2020.

[14] Roberto Rodriguez-Echeverría, Javier Luis Cánovas Izquierdo, Manuel Wimmer, and Jordi Cabot. An LSP Infrastructure to Build EMF Language Servers for Web-Deployable Model Editors. In Regina Hebig and Thorsten Berger, editors, *Proceedings of the 2nd International Workshop on Model-Driven Engineering Tools (MDE-Tools'18)*, pages 1–10, Copenhage, Denmark, October 2018. CEUR.

[15] 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.

[16] Edoardo Vacchi and Walter Cazzola. Neverlang: A Framework for Feature-Oriented Language Development. *Computer Languages, Systems & Structures*, 43(3):1–40, October 2015.

[17] Edoardo Vacchi, Walter Cazzola, Benoît Combemale, and Mathieu Acher. Automating Variability Model Inference for Component-Based Language Implementations. In Patrick Heymans and Julia Rubin, editors, *Proceedings of the 18th International Software Product Line Conference (SPLC'14)*, pages 167–176, Florence, Italy, 15th-19th of September 2014. ACM.

[18] Edoardo Vacchi, Diego Mathias Olivares, Albert Shaqiri, and Walter Cazzola. Neverlang 2: A Framework for Modular Language Implementation. In *Proceedings of the 13th International Conference on Modularity (Modularity'14)*, pages 23–26, Lugano, Switzerland, 22nd-25th of April 2014. ACM.

[19] Christian Wende, Nils Thieme, and Steffen Zschaler. A Role-Based Approach towards Modular Language Engineering. In Mark van den Brand, Dragan Gašević, and Jeff Gray, editors, *Proceedings of the 2nd International Conference on Software Language Engineering (SLE'09)*, Lecture Notes in Computer Science 5969, pages 254–273, Denver, CO, USA, October 2009. Springer.

[20] Jules White, James H. Hill, Jeff Gray, Sumant Tambe, Aniruddha Gokhale, and Douglas C. Schmidt. Improving Domain-specific Language Reuse with Software Product-Line Configuration Techniques. *IEEE Software*, 26(4):47–53, July-August 2009.

Sunday 7th July, 2024