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

COMPUTER SCIENCE DEPARTMENT • ADAPT-LAB

Academic Year 2023-2024

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.

The introduction of LSP marked a significant paradigm shift by decoupling language-specific functionalities from IDEs. This modular and scalable approach allows developers to leverage sophisticated language services without being restricted to a particular IDE. LSP facilitates a standardized communication protocol between language servers and IDEs, ensuring a uniform and streamlined development experience across various platforms. By utilizing JSON-RPC for its communication, LSP ensures that language features are implemented consistently, enhancing the overall efficiency and portability of development tools.

3 Work Objectives

The primary objectives of this work are to explore and elucidate the transformative impact of the Language Server Protocol on software development. Specifically, the goals are:

1. **Examine the foundational concepts and technologies underlying LSP**: This includes understanding the components of LSP, such as JSON-RPC and command specifications, and their roles in facilitating seamless integration between language servers and IDEs.
2. **Investigate the role of compilers and language workbenches in LSP development**: This involves exploring how compilers and language workbenches can be leveraged to integrate language-specific features into LSP, enabling the creation of robust and efficient language servers.
3. **Analyze the benefits and challenges of DSLs**: This includes examining the advantages of Domain-Specific Languages (DSLs) in the context of LSP, as well

as the challenges associated with designing and implementing DSLs for specific domains.

4. **Explore static analysis and type systems**: Delve into the theoretical and practical aspects of these technologies and their integration into language servers to enhance code quality and developer productivity.
5. **Assess the impact of modular architectures in software and language 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**: Investigate how to reduce the number of combinations ($\mathbf{L} + \mathbf{E}$ where $\mathbf{L}$ is the number of languages and $\mathbf{E}$ is the number of editors) to support LSP for $\mathbf{L}$ languages, moving towards a more modular approach ($\mathbf{L} \times 1$) 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

This thesis undertakes a comprehensive exploration of critical aspects in modern software development, with a primary focus on advancing language server protocols (LSP) and refining type systems to enhance programming language capabilities.

The Language Server Protocol (LSP) serves as a cornerstone in facilitating seamless communication between integrated development environments (IDEs) and language servers. Operating on the JSON-RPC standard, LSP defines essential commands and methods crucial for efficient source code analysis and manipulation. This protocol not only standardizes the interactions between IDEs and language servers but also supports a wide array of programming languages, thereby improving developer productivity and software quality.

Language workbenches (LWs) are pivotal tools examined in this thesis, designed to facilitate the creation and customization of domain-specific languages (DSLs). Some LWs promote modularization and composability, allowing for the efficient reuse of language components across different projects and domains.

A portion of the thesis is dedicated to exploring the theoretical foundations and practical implementations of type systems. Type systems are fundamental to program analysis and verification, ensuring type safety and enhancing code reliability. Theoretical aspects, including type theory as a logic for reasoning about program behavior, are discussed in-depth. Practical implementations such as type inference mechanisms, which automate the deduction of data types in programs, are also examined for their role in improving developer productivity and code readability.

DSLs are investigated for their ability to streamline software development by providing specialized syntax and semantics tailored to specific problem domains. Internal

and external DSLs are compared, highlighting their respective advantages in enhancing code maintainability and expressiveness. By enabling developers to write code that closely aligns with domain-specific concepts, DSLs contribute to reducing development time and minimizing errors in complex software systems.

In addition, we reduce the number of combinations ($\mathbf{L} + \mathbf{E}$) required to support LSP for **L** languages, moving towards a more modular approach ($\mathbf{L} \times 1$) to streamline LSP development. We demonstrated that generating the **client** $\mathbf{L} \times 1$ is feasible.

The thesis also delves into software and language product lines, focusing on strategies for managing feature variability across different product variants. By employing configurable feature models and systematic configuration management techniques, organizations can effectively tailor software products to meet diverse customer requirements while maintaining code coherence and integrity.

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 that facilitate the creation and customization of domain-specific languages (DSLs) 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.
- **Software Product Lines (SPLs)**: A methodology for managing feature variability across software product variants.

6 Competences and achievements

Thanks to this work, we are writing a paper for the *Journal of Systems and Software* called **Code More to Code Less: Streamlining LSP Development for Language Families**.

During the course of this research, significant competences and achievements have been attained, primarily highlighted by the development of a robust software library comprising approximately 10000 lines of Java code. This library was meticulously crafted to enhance the functionality and interoperability of language server protocols (LSP) within integrated development environments (IDEs). 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.

- 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.
- 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.

These competences and achievements underscore the dedication to advancing the state of the art in software language engineering, with a specific focus on enhancing LSP functionality and promoting effective integration within modern IDE environments. The developed library stands as a testament to the research's commitment to excellence and innovation in software development tools and methodologies.

Bibliography

[1] Sven Apel, Alexander von Thein, Philipp Wendler, Armin Größlinger, and Firk Beyer. Strategies for Product-Line Verification: Case Studies and Experiments. In Betty H. Chang and Klaus Pohl, editors, *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*, pages 482–491, San Francisco, CA, USA, may 2013. IEEE.

[2] 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.

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

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

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

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

[7] 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.

[8] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged Configuration Using Feature Models. In David Weiss and Rob van Ommering, editors, *Proceedings of the 3rd International Conference on Software Product-Line (SPLC'04)*, Lecture Notes in Computer Science 3154, pages 266–283, Boston, MA, USA, August-September 2004. Springer.

[9] 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.

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

[11] 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.

[12] 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.

[13] Thomas Kühn, Walter Cazzola, and Diego Mathias Olivares. Choosy and Picky: Configuration of Language Product Lines. In Goetz Botterweck and Jules White, ed-

itors, *Proceedings of the 19th International Software Product Line Conference (SPLC'15)*, pages 71–80, Nashville, TN, USA, 20th-24th of July 2015. ACM.

[14] 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.

[15] 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.

[16] Christian Prehofer. Feature-Oriented Programming: A New Way of Object Composition. *Concurency and Computation: Practice and Experience*, 13(6):465–501, May 2001.

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.