Toward a Modular Approach for Type Systems and LSP Generation

Federico Cristiano Bruzzone
Id. Number: 27427A

MSc in Computer Science

Relatore: Prof. Walter Cazzola
Corelatore: Dr. Luca Favalli

UNIVERSITÀ DEGLI STUDI DI MILANO
Computer Science Department
ADAPT-Lab

Academic Year 2023-2024

COMPUTER SCIENCE DEPARTMENT • ADAPT-LAB

1 Ente presso cui è stato svolto il lavoro di stage

Il mio tirocinio si è svolto presso l'**ADAPT-Lab** dell'Università degli Studi di Milano, sotto la supervisione del Prof. Walter **Cazzola** e del Dr. Luca **Favalli**. Il laboratorio fa parte del Dipartimento di Informatica ed è focalizzato sulla ricerca nel campo dei linguaggi di programmazione e dell'ingegneria del software.

2 Contesto iniziale

Il panorama dello sviluppo software sta subendo una rapida trasformazione, guidato dalla crescente complessità e diversità dei linguaggi di programmazione e degli strumenti. Tra i principali progressi, il **Language Server Protocol** (LSP) è emerso come una tecnologia cruciale, ridefinendo fondamentalmente il modo in cui gli sviluppatori interagiscono con i loro ambienti di sviluppo. Prima dell'avvento di LSP, ogni Integrated Development Environment (IDE) doveva implementare il proprio set di funzionalità specifiche del linguaggio, come il completamento automatico, il controllo degli errori e il refactoring. Questo approccio spesso risultava in un'esperienza di sviluppo *frammentata* e *incoerente*, richiedendo un notevole sforzo per mantenere e aggiornare queste funzionalità nei diversi ambienti. Grazie alla loro natura modulare ed estensibile, i **Language Workbenches** (LWs) avrebbero potuto essere utilizzati per semplificare lo sviluppo di sistemi di tipi e funzionalità specifiche del linguaggio.

3 Obiettivi del lavoro

Questa sezione delinea gli obiettivi principali dello studio, concentrandosi sui concetti fondamentali e le tecnologie dietro il **Language Server Protocol** (LSP). L'accento è posto su **analisi statica**, **Type Systems** (TSs) e l'integrazione di **compilatori** e **language workbenches** (LWs), insieme ai benefici e alle sfide dei **domain-specific languages** (DSLs) e delle **architetture modulari**.

1. *Esaminare i concetti fondamentali e le tecnologie alla base del **LSP***: Questo include la comprensione dei componenti del LSP, come JSON-RPC e le specifiche dei comandi.
2. *Esplorare l'analisi statica e i **TSs***: Approfondire gli aspetti teorici e pratici di queste tecnologie e la loro integrazione nei language server per migliorare la qualità del codice e la produttività degli sviluppatori.
3. *Indagare il ruolo dei **compilatori** e dei **LWs** nello sviluppo di LSP*: Questo comporta l'esplorazione di come le fasi del compilatore e le language workbenches possano essere sfruttati per integrare LSP.
4. *Analizzare i benefici e le sfide dei **DSLs***: Questo include l'esame dei vantaggi dei DSLs nel contesto di LSP, nonché le sfide associate alla progettazione e implementazione dei DSLs nel dominio dei **TSs**.

5. *Valutare l'impatto delle architetture modulari nello sviluppo dei **TSs** e di **LSP***: Valutare come la modularità, promossa da LSP e dagli strumenti correlati, supporti lo sviluppo efficiente e la manutenzione dei linguaggi di programmazione e delle loro funzionalità.
6. *Ridurre il numero di combinazioni per supportare* **L** *linguaggi e* **E** *editors*: Indagare come la modularità potrebbe ridurre il numero di combinazioni, passando da $\mathbf{L} + \mathbf{E}$ a $\mathbf{L} \times 1$ per semplificare lo sviluppo di LSP.
7. *Dimostrare implementazioni pratiche e **casi di studio***: Fornire esempi dettagliati e casi di studio per illustrare le applicazioni reali e i benefici dei concetti e delle tecnologie proposti.

4 Descrizione lavoro svolto

Il lavoro svolto in questo studio ha comportato un'esplorazione completa e un'implementazione pratica dei concetti fondamentali e delle tecnologie alla base del **Language Server Protocol** (LSP). L'obiettivo principale era generare l'integrazione e le funzionalità di LSP per ogni linguaggio possibile scritto in Neverlang per creare un'esperienza di sviluppo più coesa ed efficiente.

Inizialmente, è stata condotta un'analisi *approfondita* dei componenti del LSP, sottolineando il ruolo di JSON-RPC come protocollo di comunicazione e le varie specifiche dei comandi che definiscono le interazioni tra IDEs e language servers. Questa comprensione fondamentale era cruciale per sviluppare soluzioni robuste che potessero integrarsi senza problemi nei diversi ambienti di sviluppo.

La ricerca ha anche esplorato l'**analisi statica** e i **sistemi di tipi**, esaminando sia i framework teorici che le applicazioni pratiche. Queste indagini sono state fondamentali per incorporare controlli avanzati della qualità del codice e miglioramenti della produttività degli sviluppatori nei language servers. Integrando questi elementi, il progetto mirava a fornire feedback in tempo reale e rilevamento degli errori, migliorando significativamente l'esperienza di codifica.

Una parte significativa del lavoro si è concentrata sull'interazione tra **compilatori** e **language workbenches** (LWs) nel contesto dello sviluppo di LSP. Lo studio ha esaminato come diverse fasi delle operazioni del compilatore potessero essere sfruttate per supportare le funzionalità del LSP, garantendo che le caratteristiche specifiche del linguaggio fossero implementate e mantenute accuratamente. Le language workbenches, con le loro architetture modulari ed estensibili, sono stati utilizzati per semplificare la creazione e la personalizzazione di queste funzionalità.

La ricerca ha inoltre indagato i benefici e le sfide associati ai **domain-specific languages** (DSLs). Progettando e implementando DSLs su misura per domini specifici dei sistemi di tipi, lo studio mirava a dimostrare come questi linguaggi specializzati potessero migliorare la produttività degli sviluppatori e la qualità del codice. Sono state affrontate anche le sfide della progettazione e implementazione dei DSLs, fornendo approfondimenti su strategie efficaci per superare questi ostacoli.

La **modularità** è stata un tema chiave per tutto il progetto, con un forte accento su come le **architetture modulari** potessero supportare lo sviluppo e la manutenzione efficienti dei linguaggi di programmazione e delle loro funzionalità. Questo approccio modulare non solo ha facilitato il riutilizzo e la scalabilità del codice, ma ha anche svolto un ruolo cruciale nella riduzione del numero di combinazioni necessarie per supportare più linguaggi, semplificando così lo sviluppo di LSP. Infatti, riduciamo il numero di combinazioni ($\mathbf{L} + \mathbf{E}$) richieste per supportare LSP per **L** linguaggi e **E** editors a $\mathbf{L} \times 1$ combinazioni, generando i **client**.

Per illustrare le applicazioni pratiche e i benefici dei concetti e delle tecnologie proposti, sono stati forniti dettagliati casi di studio e implementazioni pratiche (ad esempio, l'integrazione di LSP e del sistema di tipi per Neverlang e Simplelang). Questi esempi hanno mostrato l'impatto reale della ricerca e hanno evidenziato il potenziale per migliorare i flussi di lavoro dello sviluppo software attraverso capacità avanzate dei language server.

5 Tecnologie coinvolte

Il lavoro ha coinvolto le seguenti tecnologie:

- **Language Server Protocol (LSP)**: Un protocollo di comunicazione standardizzato che consente l'integrazione di funzionalità specifiche del linguaggio negli IDE.
- **JSON-RPC**: Un protocollo leggero di chiamata di procedura remota che serve come base per la comunicazione LSP.
- **Compilatori e Language Workbenches (LWs)**: Strumenti e framework utilizzati per analizzare, trasformare e generare codice, nonché per sviluppare linguaggi specifici del dominio e funzionalità del linguaggio.
- **Sistemi di Tipi**: Framework teorici e implementazioni pratiche che governano i tipi di dati e le loro interazioni nei linguaggi di programmazione.
- **Domain-Specific Languages (DSLs)**: Linguaggi specializzati progettati per affrontare specifici domini di problemi e migliorare la produttività degli sviluppatori.
- **LSP4J**: Una libreria Java che fornisce supporto per l'implementazione di language server utilizzando il LSP.
- **Software Product Lines (SPLs)**: Una metodologia per gestire la variabilità delle funzionalità tra le varianti di prodotti software.
- **Linguaggi di Programmazione**: Inclusi Java, Neverlang, TypeScript, JavaScript e altri linguaggi.

6 Competenze e risultati raggiunti

Stiamo scrivendo un articolo per *Journal of Systems and Software* intitolato "**Code Less to Code More**: Streamlining LSP Development for Language Families".

Inoltre, nel corso di questa ricerca sono state acquisite competenze significative e raggiunti risultati importanti. Le competenze chiave e i risultati includono:

- *Sviluppo di una libreria*: È stata progettata e implementata una libreria software completa, sfruttando il linguaggio di programmazione Java per racchiudere le funzionalità essenziali richieste per un'integrazione efficace di LSP. Questa libreria comprende moduli per le specifiche dei comandi, la comunicazione JSON-RPC, le implementazioni dei metodi chiave e capacità avanzate di analisi del codice sorgente per i sistemi di tipi. La libreria comprende circa 10000 righe di codice.
- *Miglioramento dello sviluppo di LSP*: Concentrandosi sull'estensione delle capacità di LSP, la libreria consente una comunicazione senza soluzione di continuità tra gli IDE e i language server, favorendo una migliore navigazione del codice, evidenziazione della sintassi, completamento del codice e rilevamento degli errori in tempo reale. Questi miglioramenti elevano significativamente l'esperienza di sviluppo fornendo agli sviluppatori strumenti potenti per ottimizzare i flussi di lavoro dello sviluppo software.
- *Design modulare*: La libreria è stata progettata con un approccio modulare, facilitando l'integrazione e la scalabilità nei diversi ambienti IDE e linguaggi di programmazione. Questa architettura modulare promuove la riusabilità del codice, l'efficienza della manutenzione e l'adattabilità ai requisiti software in evoluzione.
- *Integrazione del sistema di tipi*: Integrale alla funzionalità della libreria è la sua robusta integrazione del sistema di tipi, che migliora le capacità di analisi e verifica del programma. L'implementazione include meccanismi sofisticati per l'inferenza dei tipi, la gestione delle tabelle dei simboli e la risoluzione degli scope, garantendo una gestione accurata ed efficiente dei costrutti specifici del linguaggio.
- *Un nuovo DSL per i sistemi di tipi e l'implementazione di LSP*: È stato progettato e implementato un nuovo DSL per supportare la definizione dei sistemi di tipi e delle funzionalità di LSP. Questo DSL fornisce un'astrazione ad alto livello per specificare i costrutti e le interazioni specifiche del linguaggio, consentendo uno sviluppo rapido e una personalizzazione dei sistemi di tipi e delle funzionalità del linguaggio.
- *Contributi all'ingegneria dei linguaggi*: Attraverso lo sviluppo di questa libreria, sono stati fatti significativi contributi al campo dell'ingegneria dei linguaggi. Avanzando le capacità di LSP e promuovendo le migliori pratiche nella progettazione e implementazione dei linguaggi software, la ricerca sottolinea il suo impegno a migliorare le pratiche di sviluppo software e a promuovere l'innovazione negli strumenti linguistici.

Bibliografia

[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. SPLLSP: 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.