Universal Language Server Protocol and Debugger Adapter Protocol for Modular Language Workbenches

Federico Bruzzone

Università degli Studi di Milano
Computer Science Department
PhD Candidate in Computer Science

22/07/2024
Cyclus 40th

The implementation of a programming language is a complex task that involves several implementation aspects, such as:

- Syntax and semantics definition
- Type system definition
- Code generation
- Error handling
- IDE support
- Documentation

The implementation of a programming language is a complex task that involves several implementation aspects, such as:

- Syntax and semantics definition
- Type system definition
- Code generation
- Error handling
- IDE support
- Documentation

It is usually done in a monolithic way with a top-down approach, where all the aspects are tightly coupled.

Problem Statement
Programming Language Implementation

The implementation of a programming language is a complex task that involves several implementation aspects, such as:

- Syntax and semantics definition
- Type system definition
- Code generation
- Error handling
- IDE support
- Documentation

It is usually done in a monolithic way with a top-down approach, where all the aspects are tightly coupled.

This makes the maintainability, extensibility and reusability of the implementation difficult.

In 2016, Microsoft in collaboration with Red Hat introduced the Language Server Protocol (LSP) and the Debugger Adapter Protocol (DAP).

LSP and DAP

In a Nutshell

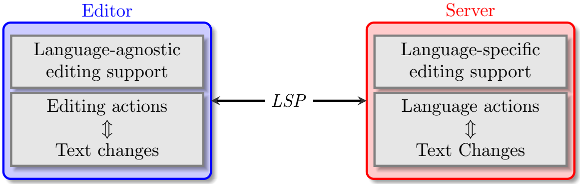
In 2016, Microsoft in collaboration with Red Hat introduced the Language Server Protocol (LSP) and the Debugger Adapter Protocol (DAP).

The LSP and DAP are JSON-RPC based protocols that allow the communication between a Language Server and an IDE.

LSP and DAP
In a Nutshell

In 2016, Microsoft in collaboration with Red Hat introduced the Language Server Protocol (LSP) and the Debugger Adapter Protocol (DAP).

The LSP and DAP are JSON-RPC based protocols that allow the communication between a Language Server and an IDE.



Intrinsic properties:

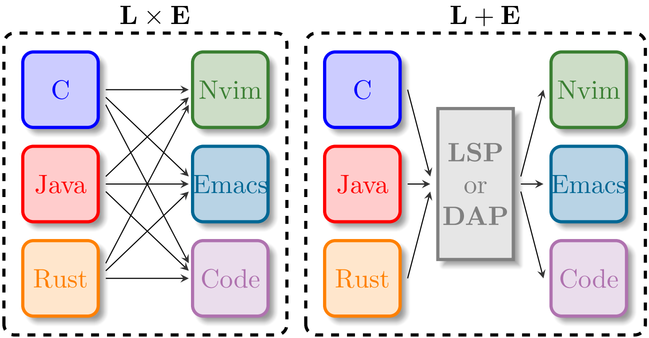
- Language-agnostic
- IDE-agnostic
- Asynchronous
- Text-based

Features:

- Diagnostics
- Hover
- Go to definition
- Find references

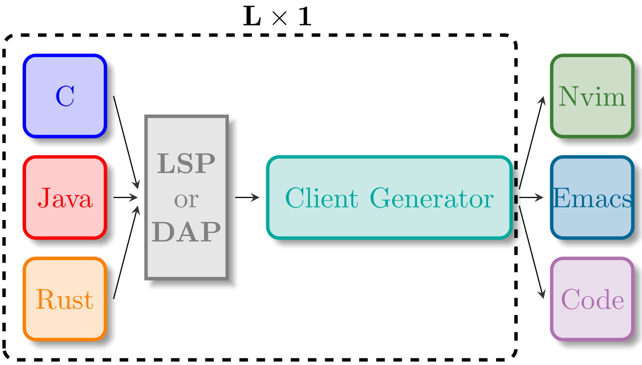
Initially implemented for Visual Studio Code, the LSP and DAP have been adopted by several IDEs and programming languages.

Initially implemented for Visual Studio Code, the LSP and DAP have been adopted by several IDEs and programming languages.



Reducing the number of combinations between Language Servers and IDEs.

Reducing the number of combinations between Language Servers and IDEs.



RO 1: Reduce to $\mathbf{L} \times \mathbf{1}$ the number of combinations to support **L** languages

Feature-Oriented Programming

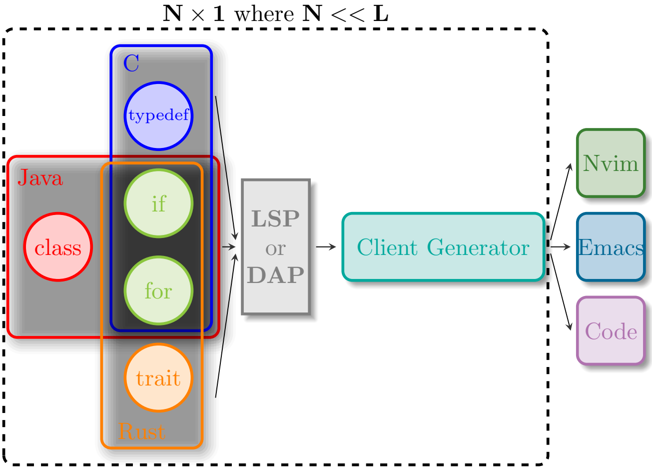
Feature-Oriented Programming (FOP) is a programming paradigm that allows the development of software product lines (SPLs).

Feature-Oriented Programming

Feature-Oriented Programming (FOP) is a programming paradigm that allows the development of software product lines (SPLs).

- Feature is a unit of functionality that satisfies a requirement.
- Feature Model is a model that represents the variability of the SPL.
- Feature Configuration is a set of features that compose a product.

RO 2: Facilitate LSP and DAP Modularization


N × 1 where N << L
C
typedef
Java
class
if
for
trait
Rust
LSP
or
DAP
Client Generator
Nvim
Emacs
Code

Language Workbenches

Language Workbenches (LWs) are tools that allow the development of programming languages, both GPLs and DSLs.

Language Workbench	Modularization Supp.	Precompiled Feature Supp.	Native IDE Gen.	LSP & DAP Gen.	LSP & DAP Mod.
JustAdd	◐	○	○	○	○
Melange	⦸	○	3rd p.	☆	☆
MontiCore	◐	◐	●	○	○
MPS	⦸	○	●	☆	☆
Rascal	○	○	●	○	○
Spoofax	⦸	◐	●	☆	☆
Xtext	○	◐	●	●	○
Neverlang	⦻	●	○	✯	✯

- ● Full support
- ○ No support
- ◐ Limited support
- ⦻ Fine-grained mod.
- ⦸ Coarse-grained mod.
- ✯ My expected contribution
- ☆ Extended contribution
- 3rd p. Third-party

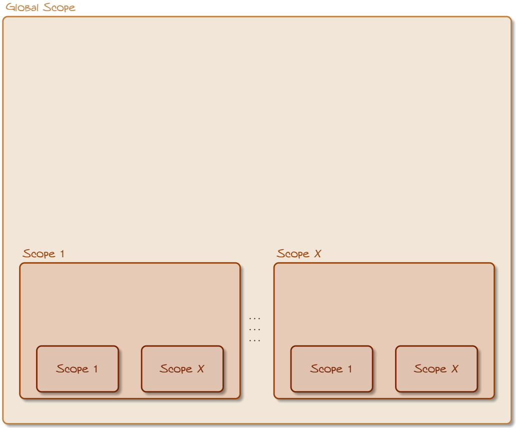
RO 3: Improve IDE and LSP Generation

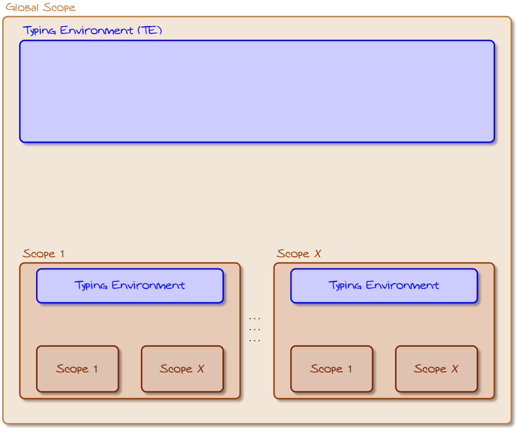
Scientific Contribution

- Methodology for whole LWs that support at least component modularization.
- Type System, LSP and DAP Modularization.
- DSL for Type System definition.
- LSP and DAP generation for Neverlang languages.
- Clients and Syntax Highlighting generation reducing the number of combinations.
- Implementation of a Java Library for Neverlang to support the type system, LSP and DAP for every language developed with Neverlang.
- 3 use cases to show the effectiveness of the methodology.

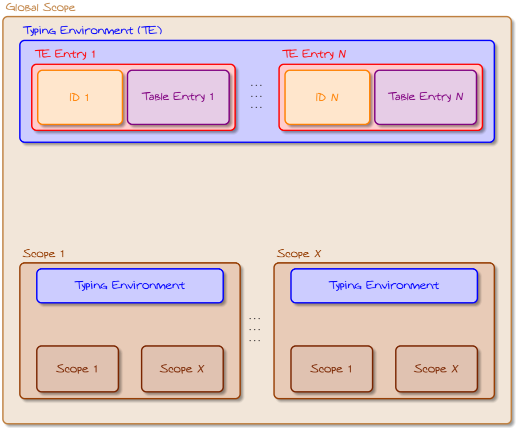
RO 4: Leverage Neverlang for LSP and DAP in LPL Development

Scientific Contribution
Type System Components

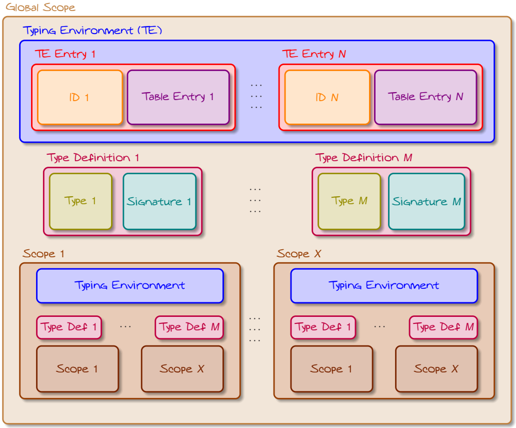



Global Scope
Typing Environment (TE)
Scope 1
Typing Environment
Scope 1
Scope X
Scope X
Typing Environment
Scope 1
Scope X

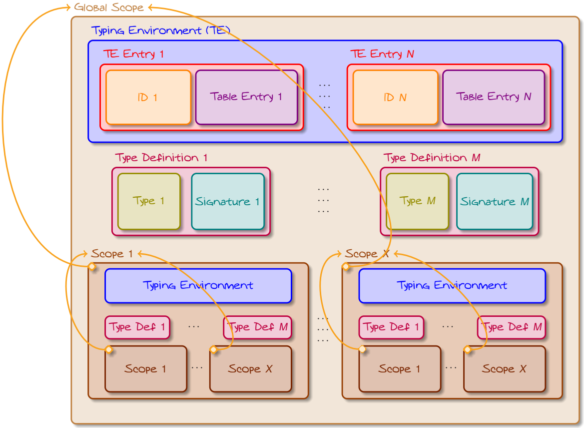
Scientific Contribution
Type System Components



Scientific Contribution
Type System Components

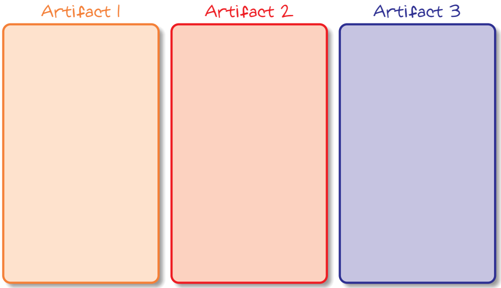


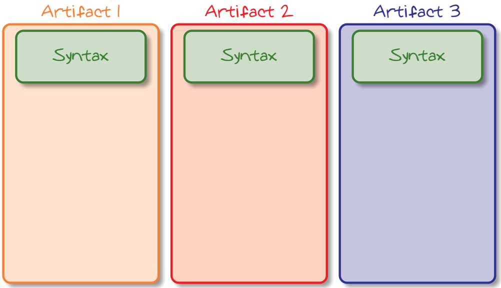
Scientific Contribution
Type System Components

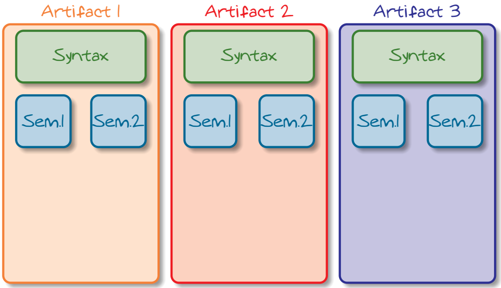


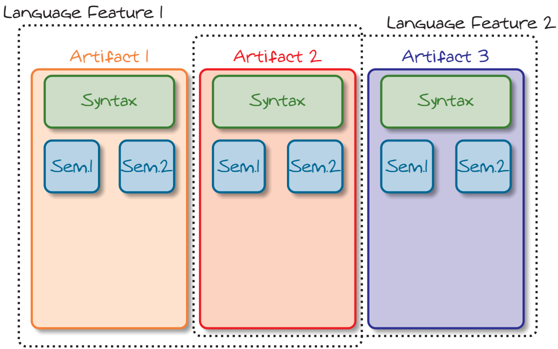
Scientific Contribution

TSs, LSP and DAP Modularization

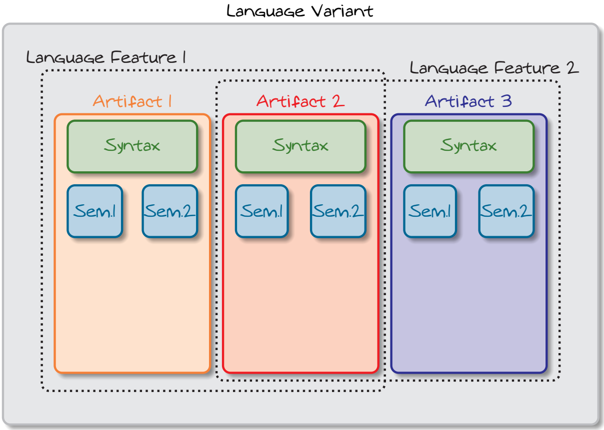



Artifact 1
Syntax
Artifact 2
Syntax
Artifact 3
Syntax


Artifact 1
Syntax
Sem.1
Sem.2
Artifact 2
Syntax
Sem.1
Sem.2
Artifact 3
Syntax
Sem.1
Sem.2

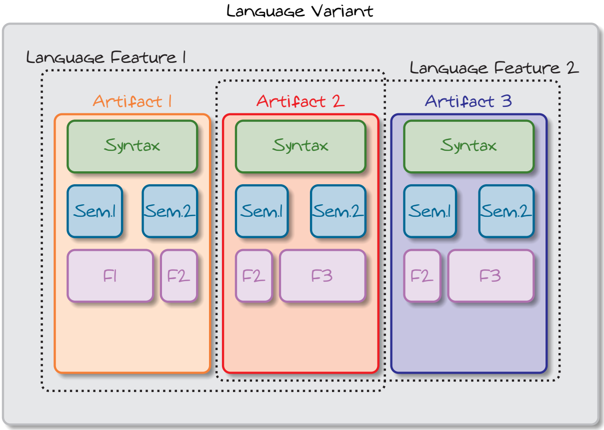


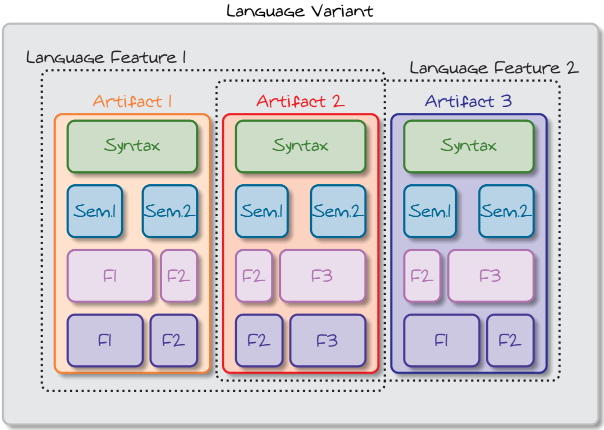
Language Feature 1
Language Feature 2
Artifact 1
Artifact 2
Artifact 3
Syntax
Sem.1
Sem.2
Syntax
Sem.1
Sem.2
Syntax
Sem.1
Sem.2

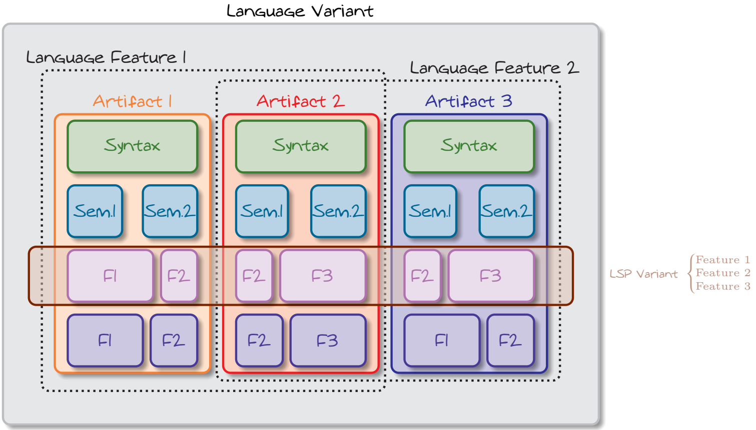


Language Variant
Language Feature 1
Language Feature 2
Artifact 1
Artifact 2
Artifact 3
Syntax
Syntax
Syntax
Sem.1
Sem.2
Sem.1
Sem.2
Sem.1
Sem.2

Scientific Contribution
TSs, LSP and DAP Modularization

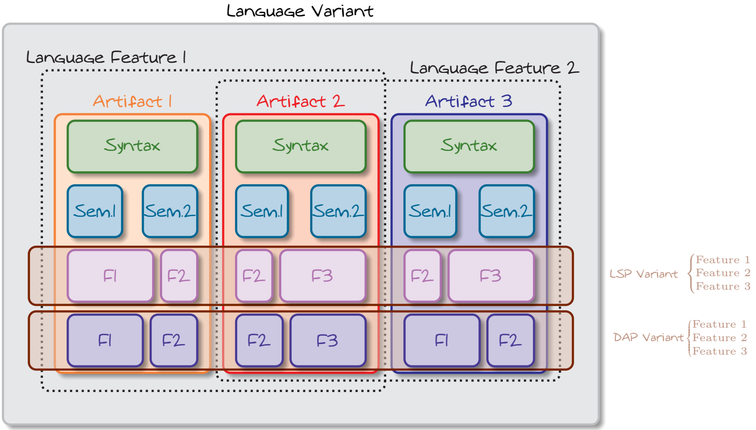



Language Variant
Language Feature 1
Language Feature 2
Artifact 1
Artifact 2
Artifact 3
Syntax
Syntax
Syntax
Sem.1
Sem.2
Sem.1
Sem.2
Sem.1
Sem.2
F1
F2
F2
F3
F2
F3
F1
F2
F2
F3
F1
F2



Language Variant
Language Feature 1
Language Feature 2
Artifact 1
Artifact 2
Artifact 3
Syntax
Syntax
Syntax
Sem.1
Sem.2
Sem.1
Sem.2
Sem.1
Sem.2
F1
F2
F2
F3
F2
F3
F1
F2
F2
F3
F1
F2
LSP Variant
Feature 1
Feature 2
Feature 3

Scientific Contribution
TSs, LSP and DAP Modularization



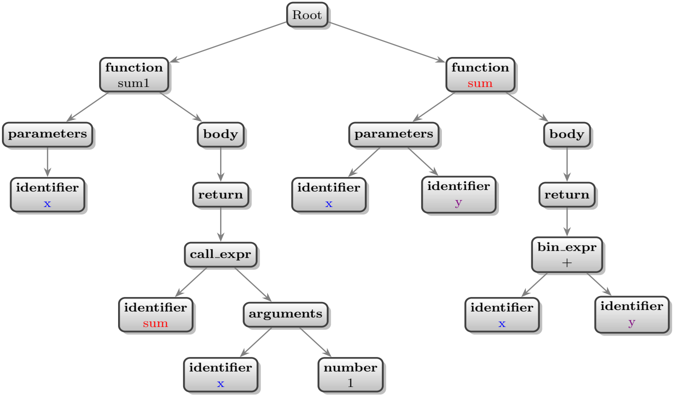
- We are writing an article (Code Less to Code More) to be submitted to JSS.
- Propose a feasibility study for the methodology.
- We prototyped the reduction of combinations.
- We prototyped the modularization of the type system.

Thanks for your attention!

Type Checking and Type Inference

```
function sum1(x) {
    return sum(x, 1);
}

function sum(x, y) {
    return x + y;
}
```

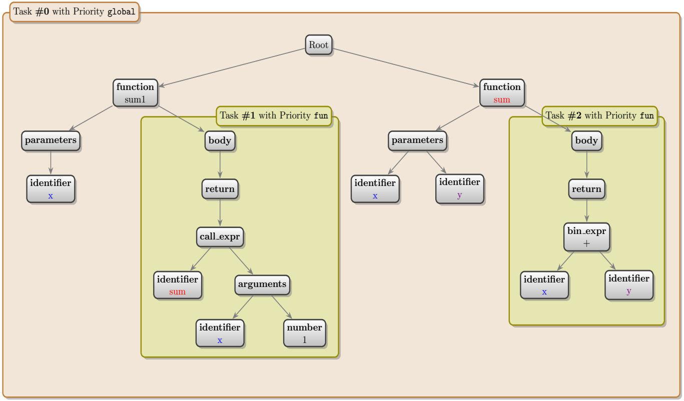


Type Checking and Type Inference


```
function sum1(x) {
    return sum(x, 1);
}

function sum(x, y) {
    return x + y;
}
```

- Compilation Unit
- Compilation Unit Task
- Compilation Helper



Software Product Lines

Since 1990s, researchers have been working on the concept of Software Product Lines (SPLs) to move towards a more modular world.

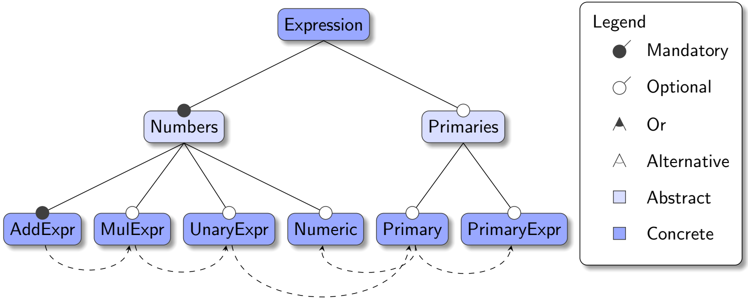
Software Product Lines

Since 1990s, researchers have been working on the concept of Software Product Lines (SPLs) to move towards a more modular world.

- SPLs defines a family of software products.
- SPLs is described by a Feature Model.
- A Feature Model describes the variability of the software.
- SPL variants are generated by selecting a set of features.
- A feature (or artifact) is a first-class entity in SPLs.

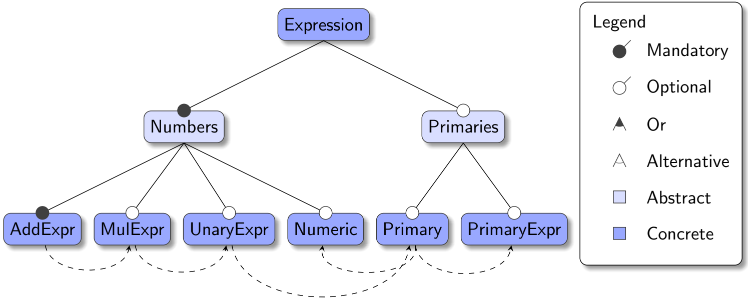
Language Product Lines

Applying the concept of SPLs to programming languages, we obtain the concept of Language Product Lines (LPLs).



Language Product Lines

Applying the concept of SPLs to programming languages, we obtain the concept of Language Product Lines (LPLs).



Some achievements:

- Bottom-up approach to language implementation
- Reusability of language artifacts
- Multiple variants of the same language
- Language Workbenches come to the rescue

Research Objective 1

RO 1: Reduce to $\mathbf{L} \times \mathbf{1}$ the number of combinations to support $\mathbf{L}$ languages

RQ 1.1: How can IDE generation be improved to support LSP and DAP?

RQ 1.2: What are the key challenges in generating LSP and DAP for different programming languages?

RQ 1.3: How can a universal LSP and DAP be developed to support multiple languages and IDEs?

Research Objective 2

RO 2: Facilitate LSP and DAP Modularization

- RQ 2.1: How can LSP and DAP modularization be facilitated in language workbenches?
- RQ 2.2: What are the key challenges in modularizing LSP and DAP for different programming languages?
- RQ 2.3: How can LSP and DAP modularization be integrated with existing language composition and modularization features in language workbenches?

Research Objective 3

RO 3: Improve IDE and LSP Generation

RQ 3.1: How can the number of combinations required to support multiple languages be reduced to L × I?

RQ 3.2: In what ways does simplifying the development process for language support enhance efficiency?

RQ 3.3: How does reducing combinations impact the speed and effectiveness of creating language support?

Research Objective 4

RO 4: Leverage Neverlang for LSP and DAP in LPL Development

RQ 4.1: How can Neverlang's LPL development features be leveraged for creating a reusable core for LSP and DAP functionalities?

RQ 4.2: What are the key benefits of using Neverlang for LSP and DAP development in the context of LPLs?

RQ 4.3: How does leveraging Neverlang's LPL features enhance the scalability and efficiency of LSP and DAP development?

Journals and Conferences

Journals

- JSS (Journal of Systems and Software)
- TSE (IEEE Transactions on Software Engineering)
- TOSEM (ACM Transactions on Software Engineering and Methodology)
- TOPLAS (ACM Transactions on Programming Languages and Systems)

Conferences

- ICSE (International Conference on Software Engineering)
- PLDI (Programming Language Design and Implementation)
- OOPSLA (Object-Oriented Programming, Systems, Languages, and Applications)
- SLE (Software Language Engineering)

Language Workbenches and Research Groups

- JustAdd → Computer Science department of the Lund University (Lund, Sweden)
- Melange → DiverSE research team at the Institut National de Recherche en Informatique et en Automatique (INRIA) (Paris, France)
- MontiCore → Software Engineering group at the RWTH Aachen University (Aquisgrana, Germany)
- MPS → JetBrains Research (Saint Petersburg, Russia)
- Rascal → Centrum Wiskunde & Informatica (CWI) (Amsterdam, Netherlands)
- Spoofax → Delft University of Technology (Delft, Netherlands)
- Xtext → Eclipse Foundation (Ottawa, Canada)

Language Workbenches and Research Groups

- **Martin Fowler:** Renowned for his work on software development methodologies. His book "Domain-Specific Languages" is a seminal work in the field.
- **Thomas Kühn:** Known for his work on Software and Language Product Lines Engineering. Professor at the Martin Luther University Halle-Wittenberg, Germany.
- **Markus Voelter:** Known for his contributions to the development and promotion of language workbenches like JetBrains MPS.
- **Eelco Visser:** A professor at Delft University of Technology, Visser has made significant contributions to the field through his work on the Spoofax language workbench.
- **Gregor Kiczales:** Known for his work on aspect-oriented rogramming (AOP). Professor at the University of British Columbia.
- **Antonia Bertolino:** Known for her work on software testing and quality assurance.