Building Code Cities using the Language Server Protocol

Master's Thesis Presentation

Falko Galperin

Faculty 3—Mathematics & Computer Science
University of Bremen

April 25, 2025

Universität Bremen

Outline

1 Introduction

2 Implementation
- Algorithm
- Integration
- Technical Evaluation

3 User Study
- Design
- Results

4 Conclusion

Overview

- **Code Cities** can help developers understand complex software systems
    - Often limited to few languages
- The **Language Server Protocol** (LSP) specifies how *Language Servers* can provide language-specific features to IDEs
    - Many Language Servers are available

Goal

☞ Integrate LSP information into code-city implementation SEE

Research Questions

Research Question 1

How can LSP be integrated into SEE to generate code cities?

Research Question 2

What is the scalability of this integration?

Research Question 3

Are code cities a suitable means to present LSP information to developers as compared to IDEs + tables (on the dimensions of speed, accuracy, and usability)?

SEE: Graph

Attributed project graph $G = (V, E, a, s, t, \ell)$

- V: Set of nodes
- E: Set of edges
- $a : (V \times \mathcal{A}_K) \rightharpoonup \mathcal{A}_V$ assigns named attributes to nodes
- $s, t : E \to V$ denotes source/target node of each edge
- $\ell : E \to \Sigma$ for labelling edges
    - Edge label `partOf` $\in \Sigma$ induces source code hierarchy

Language Server Protocol

- Open-source specification managed by Microsoft
- JSON-RPC messages sent between Language Client and Language Server
- > 260 Language Servers, > 60 Language Clients
- Specifies **capabilities** to be used by implementers
  - Examples: Go to definition, hover, document symbols
  - We will only use (some) “read-only” capabilities

Part I: Node Synthesis

For each document…

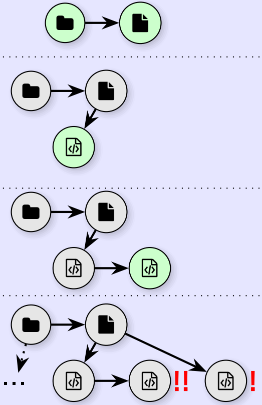
1. Add a node for the document (and its directory).

2. For each symbol in that document…

   2.1 Add a child node for the symbol.

   2.2 If there are contained symbols, go to 2.1 for each one.

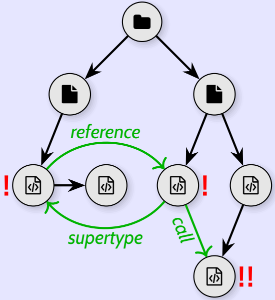
3. Retrieve diagnostics for document and attach to corresponding nodes.



Part II: Edge Synthesis

For each node…

1. Connect edge to definition, if it exists.
2. Connect edge to declaration, if it exists.
3. Connect edge to type definition, if it exists.
4. Connect edge to implementation, if it exists.
5. Connect edge to any references.
6. Connect edge to any outgoing calls using call hierarchy.
7. Connect edge to any supertypes using type hierarchy.

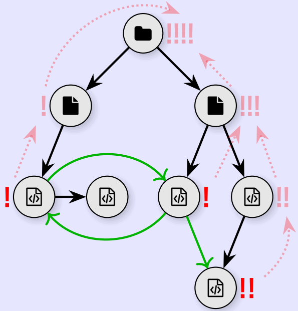


Part III: Aggregation

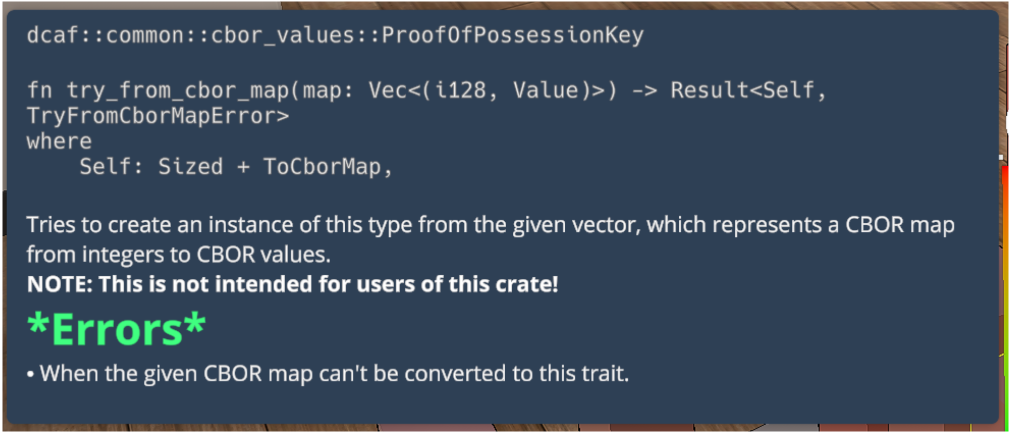
For each root node…

1. Aggregate LOC upwards.
2. Aggregate diagnostic counts upwards.

Return constructed graph.



Code Cities



```
dcaf::common::cbor_values::ProofOfPossessionKey

fn try_from_cbor_map(map: Vec<(i128, Value)>) -> Result<Self,
TryFromCborMapError>
where
    Self: Sized + ToCborMap,
```

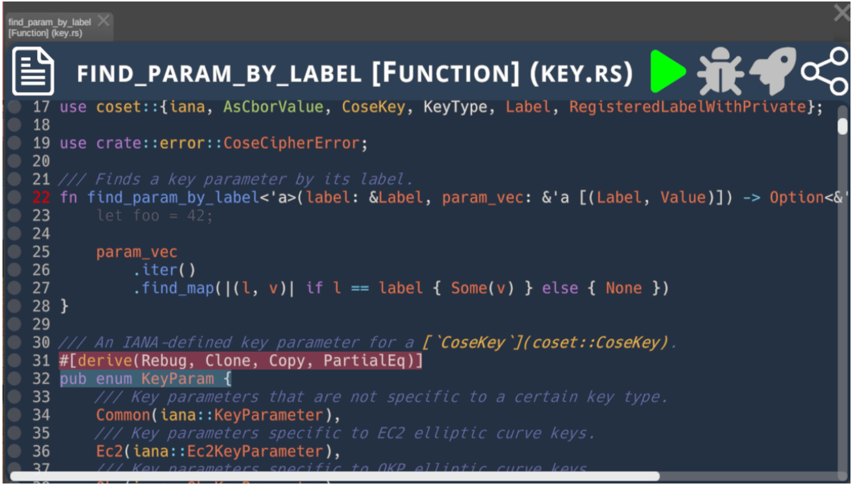
Tries to create an instance of this type from the given vector, which represents a CBOR map from integers to CBOR values.
NOTE: This is not intended for users of this crate!

Errors

- When the given CBOR map can't be converted to this trait.

Figure: A hover tooltip with information collected from the `rust-analyzer` Language Server.

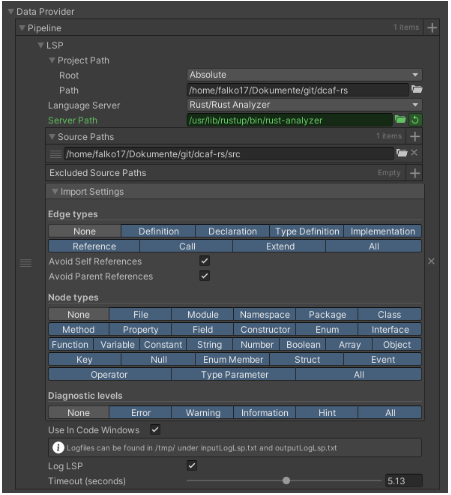
Code Windows



```
use coset::{iana, AsCborValue, CoseKey, KeyType, Label, RegisteredLabelWithPrivate};

use crate::error::CoseCipherError;

/// Finds a key parameter by its label.
fn find_param_by_label<'a>(label: &Label, param_vec: &'a [(Label, Value)]) -> Option<&'
    let foo = 42;

    param_vec
        .iter()
        .find_map(|(l, v)| if l == label { Some(v) } else { None })
}

/// An IANA-defined key parameter for a [`CoseKey`](coset::CoseKey).
#[derive(Rebug, Clone, Copy, PartialEq)]
pub enum KeyParam {
    /// Key parameters that are not specific to a certain key type.
    Common(iana::KeyParameter),
    /// Key parameters specific to EC2 elliptic curve keys.
    Ec2(iana::Ec2KeyParameter),
    /// Key parameters specific to OKP elliptic curve keys.
```

Figure: A code window with enabled LSP integration.

Generating Cities



Data Provider

Pipeline 1 items

LSP

Project Path

Root: Absolute

Path: /home/falko17/Dokumente/git/dcaf-rs

Language Server: Rust/Rust Analyzer

Server Path: /usr/lib/rustup/bin/rust-analyzer

Source Paths 1 items

/home/falko17/Dokumente/git/dcaf-rs/src

Excluded Source Paths Empty

Import Settings

Edge types

Avoid Self References ☑

Avoid Parent References ☑

Node types

Diagnostic levels

Use In Code Windows ☑

Logfiles can be found in /tmp/ under inputLogLsp.txt and outputLogLsp.txt

Log LSP ☑

Timeout (seconds) 5.13

Figure: Unity Editor UI for the LSP graph provider.

Research Question 1

How can LSP be integrated into SEE to generate code cities?

☞ Answered in previous slides

Evaluation Setup

- Ran generation algorithm on six projects in total: two per Java, Rust, LaTeX
- Measured average execution time across multiple runs
- *Only* generation algorithm included in measurement
  - E. g., no layouting or visual rendering of nodes
- Later followed up in paper submitted to ICSME 2025

Evaluation Results—Time Breakdown

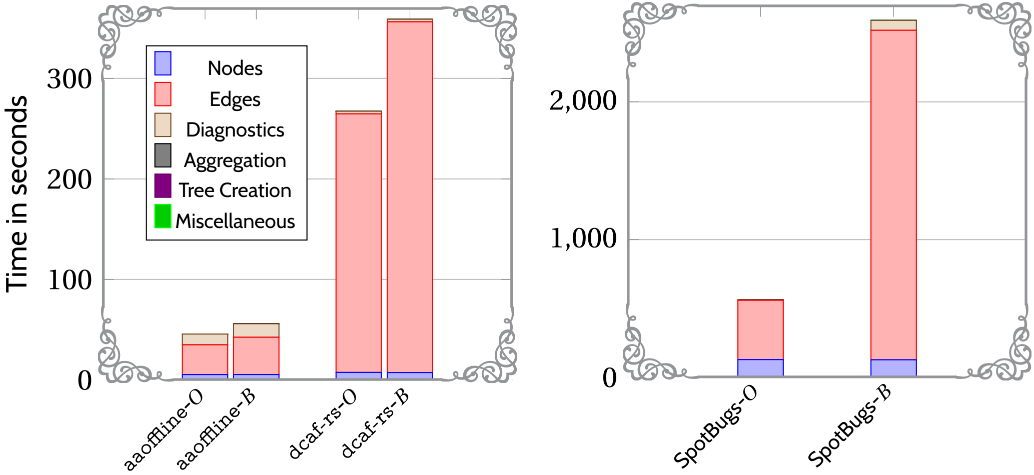


Figure: Generation time for Rust and LaTeX projects, broken down by parts of the algorithm. The suffix *O* denotes the optimized version of the algorithm, while *B* refers to the brute-force version.

Evaluation Results—Optimization Improvement

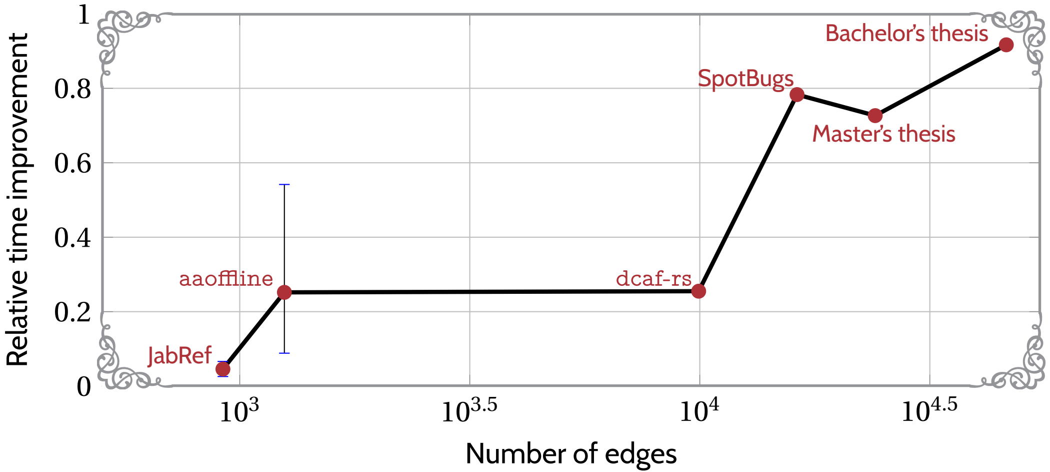


Figure: The percentage by which time is reduced when using the optimized instead of the base algorithm.

Evaluation Results—Conclusion

- Time taken only comparable within the same Language Server
- Most important metric by far: Number of edges
- Optimized variant always more performant than brute-force approach
  - $\mathcal{O}(|E| \cdot \log |V|)$ vs. $\Theta(|E| \cdot |V|)$

Research Question 2

What is the scalability of this integration?

☞ Feasible until ≈ 200 kLOC (but very dependent on configuration and Language Server)

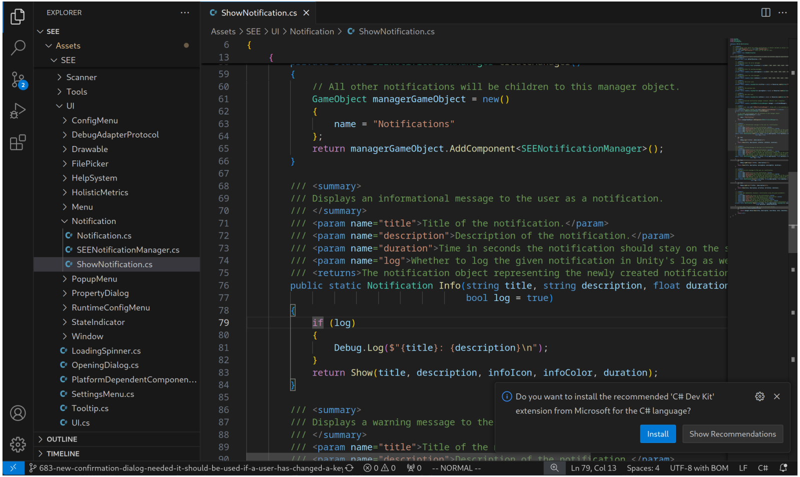
Plan

- User study comparing SEE + LSP with VSCode + LSP
- Most LSP capabilities are hard to evaluate
  - ☞ Hence, we focus on the generated city itself

We measure five dependent variables:

1. **Correctness**
2. **Speed**
3. **Usability**, differentiating between:
   - 3.1 **SUS** (post-study)
   - 3.2 **ASQ** (post-task), measuring:
     - 3.2.1 Perceived **complexity**
     - 3.2.2 Perceived **effort**

VSCode

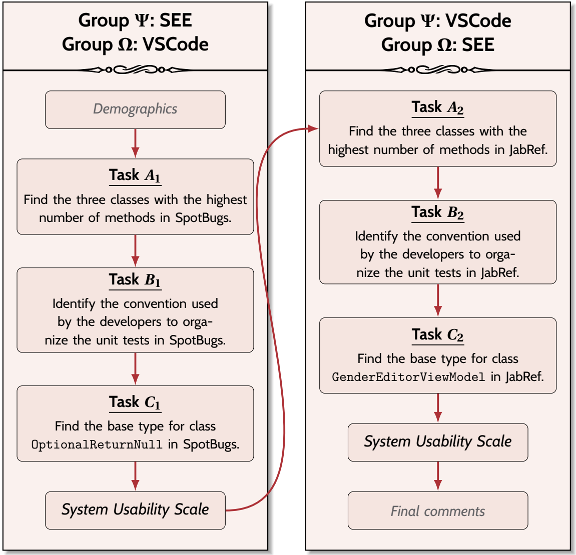


```
{
    {
        {
            // All other notifications will be children to this manager object.
            GameObject managerGameObject = new()
            {
                name = "Notifications"
            };
            return managerGameObject.AddComponent<SEENotificationManager>();
        }

        /// <summary>
        /// Displays an informational message to the user as a notification.
        /// </summary>
        /// <param name="title">Title of the notification.</param>
        /// <param name="description">Description of the notification.</param>
        /// <param name="duration">Time in seconds the notification should stay on the s
        /// <param name="log">Whether to log the given notification in Unity's log as we
        /// <returns>The notification object representing the newly created notificatior
        public static Notification Info(string title, string description, float duratior
                                        bool log = true)
        {
            if (log)
            {
                Debug.Log($"{title}: {description}\n");
            }
            return Show(title, description, infoIcon, infoColor, duration);
        }

        /// <summary>
        /// Displays a warning message to the
        /// </summary>
        /// <param name="title">Title of the
        /// <param name="description">Description of the notification.</param>
```

Do you want to install the recommended 'C# Dev Kit' extension from Microsoft for the C# language?

Figure: Screenshot of the main UI of VSCode.

Tasks


Group Ψ: SEE
Group Ω: VSCode

Demographics

Task A_1
Find the three classes with the highest number of methods in SpotBugs.

Task B_1
Identify the convention used by the developers to organize the unit tests in SpotBugs.

Task C_1
Find the base type for class `OptionalReturnNull` in SpotBugs.

System Usability Scale

Group Ψ: VSCode
Group Ω: SEE

Task A_2
Find the three classes with the highest number of methods in JabRef.

Task B_2
Identify the convention used by the developers to organize the unit tests in JabRef.

Task C_2
Find the base type for class `GenderEditorViewModel` in JabRef.

System Usability Scale

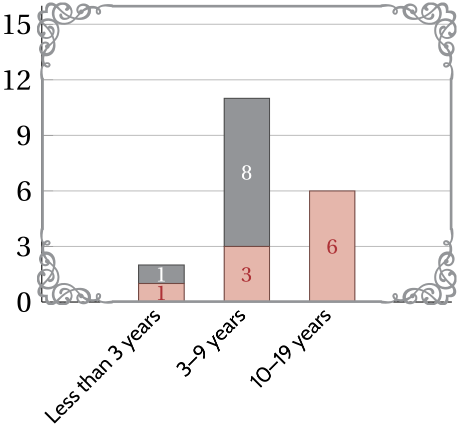
Final comments

- Within-subject study
- Three tasks per condition
- *JabRef* and *SpotBugs* used as object systems
- ASQ after each task
- SUS after each condition

Study Results

- $N = 19$ participants
- Significance level of $\alpha = 0.05$
- Used *Mann-Whitney U test* as statistical test (with some exceptions)

Figure: “How long have you been programming?”



Correctness

- Answers manually checked for correctness (to catch, e. g., misspellings)
- *Fisher's exact test* used due to binary results
- No significant differences for any of the tasks

Time

- Time measured automatically
- Only correct answers included in analysis
- Significant differences for task C (reach base class) in favor of VSCode

  ☞ Potential cause: following edges in SEE requires more navigation/interaction than just repeatedly Ctrl-clicking in VSCode

Usability: ASQ

- Two questions asked after each task
- “Cognitive” complexity: No significant differences for any task
- “Temporal” effort: Significant differences for tasks A_2 and C_1 in favor of VSCode
  - Participants indeed took longer to solve tasks A_2 and C_1 in SEE

Usability: SUS

TODO: Specify final option to see

- Ten questions asked about each system
- *Wilcoxon signed-rank test* used due to dependent samples
- Significant difference in favor of VSCode
  - SUS score falls in line with previous usability experiments involving SEE

The Effects of Experience

- We want to find correlations between independent and dependent variables
  - Especially to check if, e. g., experience may bias our results
- *Kendall's coefficient of rank correlation* τ_b used as statistical test
  - However, at 90 tests, we run into the multiple comparisons problem
  - Hence, *Benjamini-Yekutieli procedure* used to fix False Discovery Rate at 0.05
- Result: No significant correlations for any pair of variables (after FDR correction)

The Effects of Experience—Heatmap

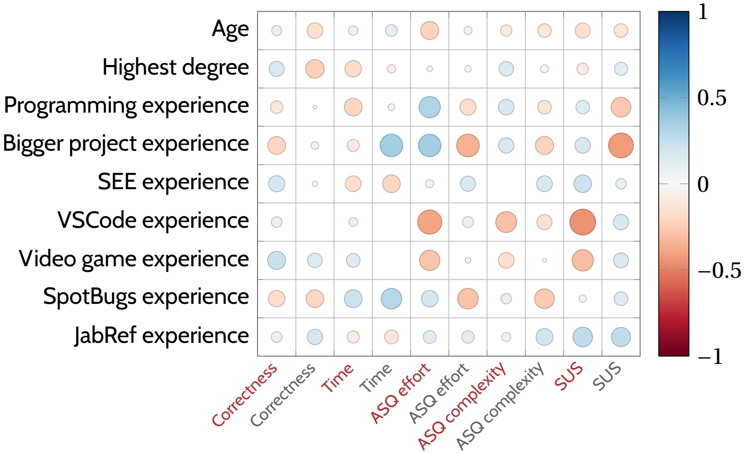


Figure: Correlations (τ_b) between independent and dependent variables. Dependent variables for SEE are marked in red and those for VSCode in gray.

Summary

Table: Significant differences between the variables, all in favor of VSCode.

Variable	A_1	B_1	C_1	A_2	B_2	C_2
Correctness	—	—	—	—	—	—
Time	—	—	$p \approx 0.0015$	—	—	$p \approx 0.0247$
ASQ: Complexity	—	—	—	—	—	—
ASQ: Effort	—	—	$p \approx 0.0142$	$p \approx 0.00531$	—	—
SUS	$p \approx 0.02116$					

Research Question 3

Are code cities a suitable means to present LSP information to developers as compared to IDEs + tables (on the dimensions of speed, accuracy, and usability)?

☞ While code cities seem suitable to present LSP information, developers work faster and prefer traditional IDEs (at least in the case of SEE vs. VSCode).

Future Work

- Improve performance to make it possible for bigger projects to run well
- Add suport for editing-related capabilities to SEE
- Use the Language Server Index Format (LSIF) to construct code cities remotely
- Utilize a registry to automatically download and set up Language Servers in SEE

Thank You!

Any questions?

`https://github.com/falko17/masterthesis`