Bachelor Thesis

A Language Server and IDE Plugin for CPAchecker

Adrian Leimeister

Software and Computational Systems Lab
LMU Munich

15.07.2020
Motivation

Powerful IDEs are available, with support for all kind of different development tools.

Options for integrating formal verification into a graphical development workflow exist, but:

- offer only rudimentary integration (Predator)
- rely on old IDE versions (CMBC/CProver)
- require separate installation of tools
- are only available for specific IDEs
Goal

Implement a CPAchecker Language Server Protocol (LSP) server and a client plugin for an IDE

Requirements:

- implement LSP server for CPAchecker
- allow CPAchecker configuration for basic use
- allow verification via CPAchecker locally
- allow verification via VerifierCloud
- visualize results in Eclipse via LSP client plugin
Software Components: Language Server Protocol (LSP)

The Problem: a Matrix

<table>
<thead>
<tr>
<th></th>
<th>Go</th>
<th>Java</th>
<th>TypeScript</th>
<th>...</th>
<th>Language N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emacs</td>
<td>Plugin 1</td>
<td>Plugin 2</td>
<td>Plugin 3</td>
<td>...</td>
<td>Plugin N</td>
</tr>
<tr>
<td>Vim</td>
<td>Plugin 1+N</td>
<td>Plugin 2+N</td>
<td>...</td>
<td>...</td>
<td>Plugin N*2</td>
</tr>
<tr>
<td>VSCod</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDE M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Plugin N*M</td>
</tr>
</tbody>
</table>

N Languages for M IDEs requires N*M Plugins!
Software Components: Language Server Protocol (LSP)

LSP: split tool support into two parts, client and server

The Solution: Clients and Servers

<table>
<thead>
<tr>
<th>Language</th>
<th>Server</th>
<th>IDE</th>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go</td>
<td>Server 1</td>
<td>Emacs</td>
<td>Client 1</td>
</tr>
<tr>
<td>Java</td>
<td>Server 2</td>
<td>Vim</td>
<td>Client 2</td>
</tr>
<tr>
<td>TypeScript</td>
<td>...</td>
<td>VSCode</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language N</td>
<td>Server N</td>
<td>IDE M</td>
<td>Client M</td>
</tr>
</tbody>
</table>

N Languages for M IDEs requires only N servers and M plugins/clients!
So, is LSP the solution to all Problems?
No:

- debugging not handled by the LSP
- server specific configuration in client

⇒ Number of Plugins required is more than \( N+M \), but reusable ”intelligent” server
Software Components: Language Server Protocol (LSP)

Example communication between IDE and language server:

User opens document
User edits document
User executes "Goto definition"
User closes document

Notification: textDocument/didOpen; Params: document

Notification: textDocument/didChange; Params: {documentURI, changes}

Notification: textDocument/publishDiagnostics; Params: Diagnostic[]

Request: textDocument/definition; Params: {documentURI, position}

Response: textDocument/definition; Result: Location

Notification: textDocument/didClose; Params: documentURI

Server publishes errors and warnings
Software Components: Eclipse IDE

- IDE with lots of features
- support for lots of tools
- written in Java
- extensible with plugins (everything is a plugin)
- basic LSP client plugin, LSP4E
Software Components: CPAchecker

- framework for formal verification
- written in Java
- extensible by implementing ”Configurable Program Analysis” (CPA) interface
- cloud verification available via API
- program converted to control flow automata (CFA)
- CFA is tested with CPAs against specification
- can be configured to produce violation witness
Implementation: Language Server

- started by language client
- communicates with client via JSON-RPC
- receives notification on file save
- requests configuration from client
- starts the configured runner
- sends result back to client
Implementation: Language Client

- based on LSP4E
- includes an installation of CPAchecker
- enables basic configuration of verification tasks
- interacts with Eclipse via extension points
- starts LSP server
- sends and receives messages to/from LSP server
- visualizes received information

---

Extension points

- content type: extension of C source file
- content type binding: open content type with C editor
- content type mapping: start LSP server when content type is opened
- ...

Adrian Leimeister (SoSy-Lab)
Implementation: Interfacing with the VerifierCloud

Communication with the VerifierCloud using HTTP requests:

- calculate file hash
- upload file to cloud
- submit run using file hash and configuration
- periodically check for completion
- download result
Implementation: Interfacing with CPAchecker

CPAchecker classes are used directly.

Main tasks for interfacing with CPAchecker:

- CPAchecker configuration creation
- message handling
- result processing
Implementation: Build Process

- built with Maven using Tycho
- Maven and Tycho have different repository types
- Eclipse plugins are OSGi bundles
- plugin dependencies are OSGi bundles
- plugin dependencies defined by OSGi manifest
- Maven Eclipse plugin projects are called ”Manifest-first”
Maven project dependencies defined by POM
”POM dependencies” not loaded by Eclipse runtime
cpachecker-lsp language server not an OSGi bundle
OSGi manifest generation possible, called ”POM-first”
Problem: ”Manifest-first” and ”POM-first” incompatible in multi-module project
Implementation: Build Process

The Solution:

- keep cpachecker-lsp as "POM dependency"
- add cpachecker-lsp build artefact to plugin output
- add cpachecker-lsp to plugin classpath
Evaluation: Method

Survey using Google Forms among students and employees at SoSy-Lab.

Gathering feedback:

- problems regarding installation process
- sufficiency of configuration options
- satisfaction with presentation of results
- ideas for improvement
Evaluation: Results

Installation:

- no problems
- caveat: requires at least Eclipse 2019-06
Evaluation: Results

Configuration:

Are the configuration options enough to cover your use case?

Votes

<table>
<thead>
<tr>
<th>Answer</th>
<th>Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>
Evaluation: Results

Presentation of Results:

Are you satisfied with the presentation of the results?

<table>
<thead>
<tr>
<th>Answer</th>
<th>Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>
Evaluation: Results

Bugs:

- wrong Java version used for starting the server ✓
- configuration corruption ✓
- confusing notification ✓
- cloud verification not working ✓
- error markers not disappearing after fixing an error ✗
- language server stopped working ✗
Evaluation: Results

Ideas for Improvement:

- improvement of error marker position ✓
- more configuration options ✓
- manual starting of verification task ✗
- per file configuration ✗
- better presentation of witnesses ✗
Installation via update site

Settings inside Eclipse
 CPAchecker LSP: Demonstration

```c
int main() {
    int i = 0;
    int a = 0;

    while (1) {
        if (i == a) {
            goto LOOPEND;
        } else {
            i++;
            a++;
        }
    }
    goto ERROR;
    if (i != a) {
        goto ERROR;
    }
}
```

CPAchecker Language Server (org.eclipse.lsp4e.LanguageServerWrapper@7caf0725)

<table>
<thead>
<tr>
<th>Log</th>
<th>Time</th>
<th>Level</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Log]</td>
<td>[2020-07-13 21:10:10] INFO</td>
<td>Verification started</td>
<td></td>
</tr>
<tr>
<td>[Log]</td>
<td>[2020-07-13 21:10:12] INFO</td>
<td>Uploading File to VerifierCloud</td>
<td></td>
</tr>
<tr>
<td>[Log]</td>
<td>[2020-07-13 21:10:12] INFO</td>
<td>Submitting run configuration to VerifierCloud</td>
<td></td>
</tr>
<tr>
<td>[Log]</td>
<td>[2020-07-13 21:10:14] INFO</td>
<td>Waiting for verification run completion</td>
<td></td>
</tr>
<tr>
<td>[Log]</td>
<td>[2020-07-13 21:10:18] INFO</td>
<td>Downloading results to temporary files</td>
<td></td>
</tr>
<tr>
<td>[Log]</td>
<td>[2020-07-13 21:10:18] INFO</td>
<td>Unpacking results</td>
<td></td>
</tr>
<tr>
<td>[Log]</td>
<td>[2020-07-13 21:10:18] INFO</td>
<td>Verification result: false</td>
<td></td>
</tr>
<tr>
<td>[Log]</td>
<td>[2020-07-13 21:10:18] SEVERE</td>
<td>Property Violation: error label in line 12</td>
<td></td>
</tr>
</tbody>
</table>
Future Work

- Language clients for more IDEs
- Better/more configuration options
- Better presentation of results
- Interactive witnesses using the Debug Adapter Protocol
Conclusion

Fulfilled goals:

- implement Language Server Protocol (LSP) server for CPAchecker: ✓
- allow CPAchecker configuration for basic use: ✓
- allow verification via CPAchecker locally: ✓
- allow verification via VerifierCloud: ✓
- visualize results in Eclipse via LSP client plugin: ✓

Additionally:

- easy installation
- easy to bring to other IDEs
Sources

https://langserver.org/
https://www.cprover.org/eclipse-plugin/
https://www.fit.vutbr.cz/research/groups/verifit/tools/predator/
https://monteverdi.informatik.uni-freiburg.de/tomcat/Website/
https://microsoft.github.io/language-server-protocol/overviews/lsp/img/language-server-sequence.png
https://projects.eclipse.org/projects/technology.lsp4e
https://www.eclipse.org/ide/
https://cpachecker.sosy-lab.org/