Format-independent Graph Vizualization with Language Server Protocol

Attila Gyén, Norbert Pataki

gyenattila@gmail.com, patakino@.elte.hu

Department of Programming Languages and Compilers
Eötvös Loránd University, Faculty of Informatics
Pázmány Péter sétány 1/C H-1117 Budapest, Hungary

Introduction

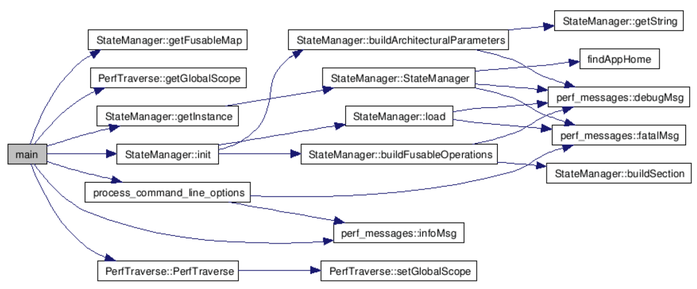
Nowadays, software lifecycles can last for years or even decades, which can lead to a number maintenance and development issues. As a software size increases, more and more developers are involved in projects [7]. Code-support tools like CodeCompass make it easier for newcomers to join projects more efficiently. To use this tool effectively, it needs to be integrated into each developer tool, IDEs [1].

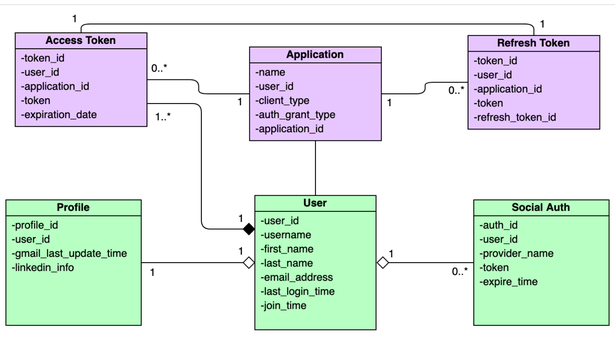
The Language Server Protocol (LSP) is a tool based on the client-server mechanism, in which the client – the development environment – can send requests to the server, which can display in the development environment – typically textual – information received in response to that programming language [3].

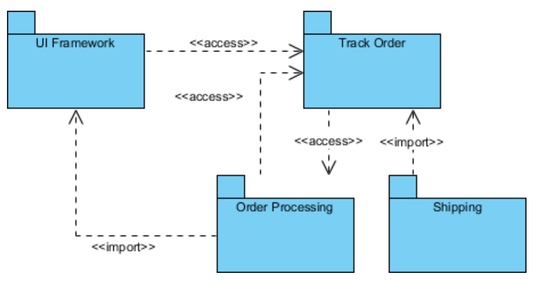
Graphs are common data structures that can be used during software development. Control-flow graphs, many UML diagrams (e.g. class diagram) are graphs. LSP currently does not support them comprehensively [4]. In this paper, we present our approach for graph vizualization in IDEs. The graphs are queried based on the source code. Different IDEs support different formats, therefore we define a construct for IDE-independency. We also present how menus and submenus can be defined with LSP.

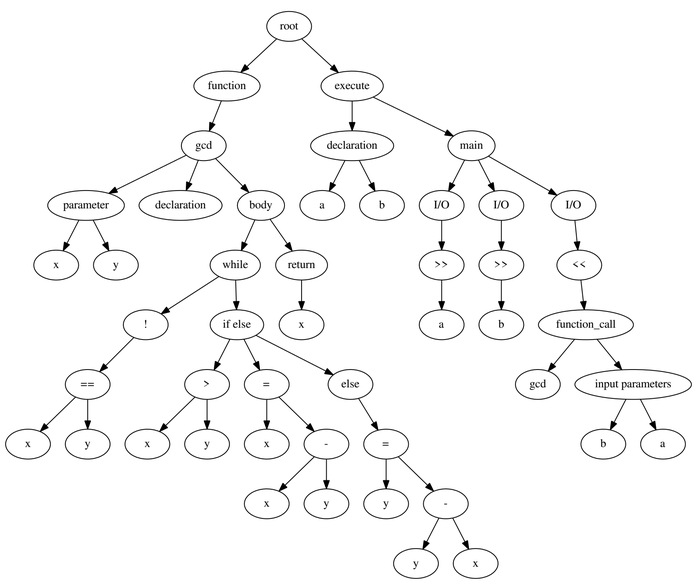
Graphs in Software Engineering

Graphs are very common data structure in software engineering. Graphs can be comprehended in a graphical way. Some examples:







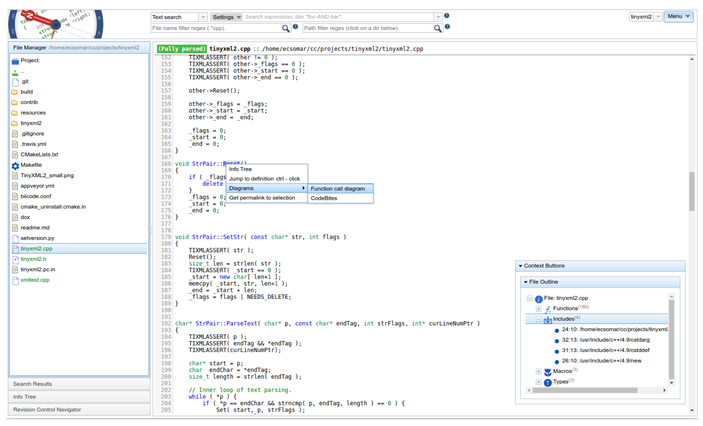


On the other hand, many graphical format should be supported because different IDEs may display different format of images:

- Scalable Vector Graphics (SVG)
- Joint Photographic Experts Group (JPEG)
- Portable Network Graphics (PNG)
- PostScript (ps)
- Portable Document Format (pdf)
- etc.

Tools - CodeCompass

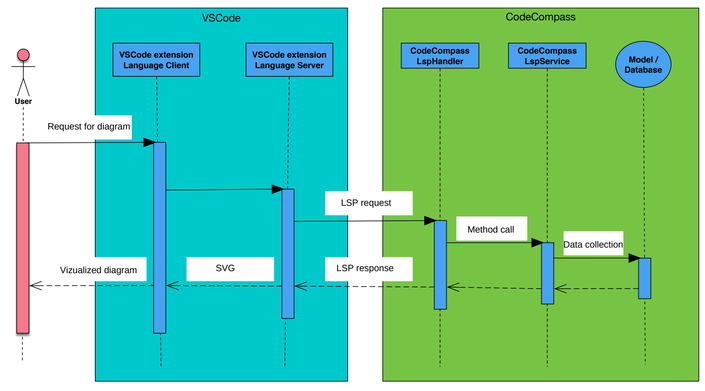
CodeCompass is an open source LLVM/Clang code comprehension tool, developed by Ericsson Ltd. in co-operation with Eötvös Loránd University to help developers to understand large software systems more easily and faster [6].



- User friendly web UI
- Fast navigation among source code elements
- Several languages supported

Our approach

The Language Server Protocol (LSP) is an open, remote procedure call protocol that commuciates between the editors/IDEs and servers that provide programming language-specific features [5]. The goal of the protocol is to allow programming language support to be implemented and distributed independently of any given editor or IDE. We use this protocol for a format-independent graph vizualization.



Because the higher-level Language Server Protocol requests cannot communicate directly with lower-level CodeCompass functions, it was necessary to create a wrapper that can combine existing functionality for LSP. This packaging service is called `LspService`.

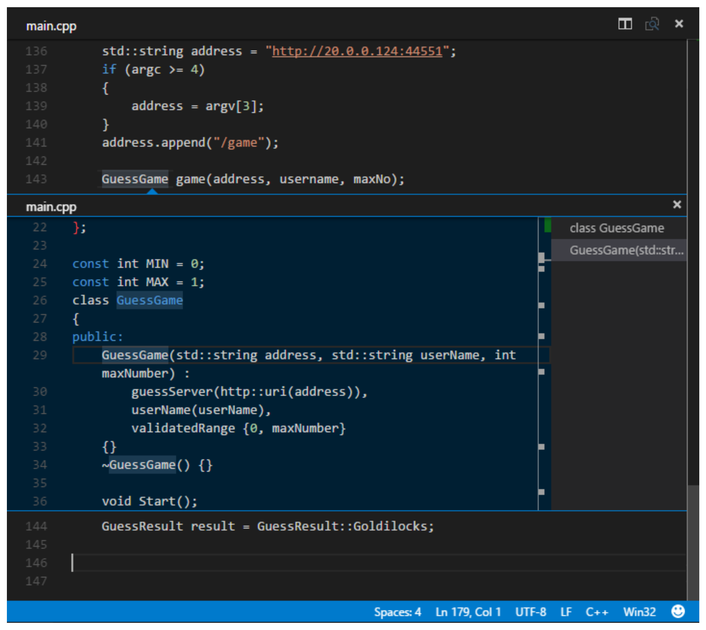
By default, the LSP does not include that feature which allows one to create a context menu at any point in the document, which one can select from among the executable commands. However, the protocol provides support for executing pre-recorded commands. Although we can insert these at a specific point in the document, this point must be fixed in advance.

Each of the IDE that support LSP has a context menu, but they are not freely expendable. The predefined functions are automatically displayed when they are implemented. However, with a little trick, protocol development – without making any changes to the original implementation - can be achieved by using an incorrect code autocompletion. This allows us to create a context menu, the content of which can also depend on the position of the current document. This can be achieved by using code-triggering characters by specifying the `context.triggerKind` and `context.triggerCharacter` properties in the CompletionParams property.

We can set any trigger character whose behavior is overridden by the contents of the trigger field. This enables us to display a list of executable commands we have written ourselves in a context-sensitive menu. We can also associate these with documentation by specifying the detail property or by allowing commands to be associated with it by specifying the command property in `CompletionItem` property.

Visual Studio Code

Visual Studio Code is open source editor developed by Microsoft. It supports many languages, operating systems and platforms [2].



Conclusion

The lifecycle of large software systems can last for years or even decades, leading to many difficulties such as maintainability and upgradeability. Due to the continuous increase in the size of software, more and more developers need to be involved in the development, for whom it is important to integrate into the development team as quickly as possible. However, this requires acquiring the relevant knowledge of the underlying codebase in the shortest possible time. Code comprehension tools like CodeCompass help obtaining this information faster and more efficiently. However it is not enough in itself to have such tools at our disposal. It is important that we can use the features provided by these tools in the most convenient way and implement them in more and more development environments. Some of the most commonly used functions are the Goto definition and the autocompletion. Conventionally, these functions needed to be reimplemented from programming language to programming language for each development tool, since each of the languages use different APIs to create the same functions. The Language Server Protocol (LSP) provides a solution to this problem. While the above functions can be implemented without the LSP – as it has been done in the past – the protocol provides a tool to standardize the interface of these functions to different development tools. The real advantage of LSP lies in standardizing communication between programming languages and development tools. We developed a simpler connection between CodeCompass and VSCode over LSP, solving problems such as running the language server and the language client on a separate server, transmitting and displaying diagrams from the language server to the client, or create context dependent menus.

References

[1] Hendrik Bünder: Decoupling Language and Editor – The Impact of the Language Server Protocol on Textual Domain-Specific Languages, in Proc. of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019), pp. 131–142.

[2] Alessando Del Sole: *Visual Studio Code Distilled: Evolved Code Editing for Windows, macOS, and Linux*, Apress, Berkely, CA, USA, 2019.

[3] Linghui Luo, Julian Dolby, Eric Bodden: *MagpieBridge: A General Approach to Integrating Static Analyses into IDEs and Editors*, in Proc. of the 33rd European Conference on Object-Oriented Programming (ECOOP 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 21(1)–21(25).

[4] Mónika Mészáros, Máté Cserép, Anett Fekete: *Delivering Comprehension Features into Source Code Editors through LSP*, in Proc. of the 2019 42nd InternationalConvention on Information and Communication Technology, Electronics and Microelectronics (MIPRO 2019), pp. 1581–1586.

[5] Daniel A. A. Pelsmaeker, Hendrik van Antwerpen, H., Eelco Visser: *Towards language-parametric semantic editor services based on declarative type system specifications*, in Proc. Companion of the 2019 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH Companion 2019), pp. 19–20.

[6] Zoltán Porkoláb, Tibor Brunner, Dániel Krupp, Márton Csordás: *CodeCompass: an open software comprehension framework for industrial usage*, in Proc. of the 26th Conference on Program Comprehension (ICPC' 18), pp. 361–369.

[7] Márk Török, Norbert Pataki: *Service Monitoring Agents for DevOps Dashboard Tool*, in Proc. of the 21th International Multi-Conference INFORMATION SOCIETY IS'2018, Volume G : Collaboration, Software and Services in Information Society, pp. 47–50.