

# Visual Mapping of Source Code to Abstract Syntax Tree

Capstone Project Final Report

Bruno Ricardo Soares Pereira de Sousa Oliveira



Bachelor's in Informatics and Computing Engineering

**U.Porto's Tutor:** Prof. Nuno Macedo  
**Company's Tutor:** Prof. João Bispo

August 2024

## Contents

<b>List of Figures</b>	<b>2</b>
<b>Glossary</b>	<b>3</b>
<b>1 Introduction</b>	<b>4</b>
1.1 Background and Scope . . . . .	4
1.2 Goals and Expected Results . . . . .	4
1.3 Report Structure . . . . .	4
<b>2 Methodology and Main Activities</b>	<b>6</b>
2.1 Used Methodology . . . . .	6
2.2 Stakeholders, Roles and Responsibilities . . . . .	6
2.3 Developed Activities . . . . .	7
2.3.1 First Weekly Meeting . . . . .	7
2.3.2 Second Weekly Meeting . . . . .	8
2.3.3 Third Weekly Meeting . . . . .	8
2.3.4 Report and Presentation Support Delivery . . . . .	9
2.3.5 Fourth Weekly Meeting . . . . .	9
2.3.6 Final Session . . . . .	9
2.3.7 Gantt Chart . . . . .	9
<b>3 Solution Development</b>	<b>10</b>
3.1 Requirements . . . . .	10
3.1.1 Functional Requirements . . . . .	10
3.1.2 Non-Functional Requirements . . . . .	10
3.2 Architecture and Technologies . . . . .	10
3.2.1 Architecture . . . . .	10
3.2.2 Clava/Backend Technologies . . . . .	11
3.2.3 Frontend Technologies . . . . .	13
3.3 Developed Solution . . . . .	13
3.3.1 API Extension . . . . .	13
3.3.2 AST and Source Code Visualization . . . . .	14
3.3.3 Mapping Between AST and Source Code . . . . .	15
3.3.4 Script Execution Control . . . . .	15
3.3.5 Syntax Highlighting . . . . .	16
3.3.6 Other Interface Features . . . . .	16
3.3.7 Integration with Other Compilers . . . . .	16
3.4 Validation . . . . .	18
<b>4 Conclusions</b>	<b>19</b>
4.1 Achieved Results . . . . .	19
4.2 Lessons Learned . . . . .	19
4.3 Future Work . . . . .	19
<b>References</b>	<b>21</b>

## List of Figures

1	Prototype's interface . . . . .	7
2	Gantt chart . . . . .	9
3	Solution architecture . . . . .	11
4	Basic usage example of the API . . . . .	13
5	Final interface design . . . . .	14
6	Highlight with mouse click . . . . .	15
7	Declaration of two variables in a single line . . . . .	17
8	Compiler-abstracted system class diagram . . . . .	17

## Glossary

**API** Application Programming Interface; set of rules or protocols that enables software applications to communicate with each other, to exchange data or perform operations [14]. 4, 6, 7, 10, 11, 12, 13, 14, 15, 18, 20

**AST** Abstract Syntax Tree; tree-like data structure that represents the hierarchical structure of the source code [3]. 4, 7, 8, 10, 13, 14, 15, 16, 18, 20

**server** Software (in the context of this report) that provides a service to a user, or client, by passing messages using a protocol over an API [4]. 10, 12, 13, 14, 18, 20

**source-to-source compiler** Also known as a source-to-source translator; compiler that transforms the source code into code in the same language or another high-level programming language [2]. 4, 13, 16

**URL** Uniform Resource Locator; text string that specifies where a resource can be found on the Internet [9]. 12, 14

# 1 Introduction

## 1.1 Background and Scope

This report aims to present and document the results and the process of my extracurricular internship, within the scope of the Capstone Project curricular unit.

The internship was developed in the context of the INESC TEC 2024 Summer Internship program. INESC TEC (Institute for Systems and Computer Engineering, Technology and Science) is a private non-profit research association, dedicated to scientific research and technological development, among other activities [27].

More specifically, this internship was carried out and organized by the SPeCS (Special-Purpose Computing Systems, Languages and Tools) research team, focused on the design and research of compilers, languages and computer architectures. The team developed the LARA framework, which provides tools and APIs useful for creating source-to-source compilers [21], and Clava, a source-to-source compiler for C/C++ (and CUDA and OpenCL), based on the LARA framework, capable of analyzing source code, converting it into an Abstract Syntax Tree (AST) and applying transformations on it, based on JavaScript scripts, to generate new C/C++ source code [24].

Before this internship, it was possible to obtain the source code and a textual representation of the program's AST using Clava. However, there was no straightforward way to associate the AST nodes with their respective code portions inside the program's source code. Consequently, there was the necessity of creating an interface that can visually perform this association, which is the aim of this internship.

## 1.2 Goals and Expected Results

The objective of this internship is to develop a web visualization tool capable of displaying the AST and the program's source code, side-by-side, such that users can easily map AST nodes to their corresponding code segments. For this effect, when the user hovers an AST node, the respective code should appear highlighted, and vice-versa. In addition, the tool must be user-friendly, intuitive and as compatible with Clava as possible (i.e. it should perform the mapping with minimal constraints).

To accomplish this, I planned on developing the solution in two parts. Firstly, I would create a simple prototype, that could perform the mapping on a basic level, with a trivial interface. Secondly, I would perfect the prototype, by improving the interface, upgrading the communication with the compiler, implementing new features and fixing bugs.

Even though these were the initial goals, the time window of this internship allowed more requirements to be tackled, which are referred to in their respective sections.

## 1.3 Report Structure

This report follows the structure below:

**1. Introduction:** The current section presents the objectives of this report, as well as the context in which the internship occurred, the problem and motivation behind this work, its expected results and an explanation of the report's structure.

**2. Methodology and main activities:** This section explains the methodology employed throughout the internship, identifies the stakeholders and other involved parties, along with their roles and responsibilities, and describes the main activities undertaken during this project, highlighting key events and deliverables.

**3. Solution development:** This section of the report discusses the project's technical elements in more depth, including key requirements, the architecture and technologies used, all the relevant functionalities of the final result, from the user's viewpoint, and the outline of the validation process, with how the solution was tested and evaluated.

**4. Conclusions:** This section finalizes the report by recapping the internships' results, reflecting on the lessons learned and proposing future improvements to the developed project.

## 2 Methodology and Main Activities

### 2.1 Used Methodology

This internship was conducted at SPeCS in a hybrid model. The other interns and I had the flexibility to work on the project remotely and communicate with our supervisors and the rest of the team through a Slack [28] channel, that belongs to the SPeCS team and to which we were invited. However, we were responsible for being present in the SPeCS room at least 1–2 days per week, to present and discuss our progress. Furthermore, our tutors recommended that we come in more frequently at the beginning of the internship, so that they could assist us more easily in the installation and configuration of Clava and adapting to the use of the compiler and the structure of the relevant projects. Despite this, I took the initiative of working on-site almost every day, which not only did I prefer, relative to working remotely, but also allowed me to ask questions and receive feedback from my tutors more effectively.

Throughout the process, we held weekly meetings in the SPeCS room for the interns to present what they had accomplished each week and to discuss the results and what should be addressed in the following week. For this purpose, we were asked to prepare a short presentation, with a duration of about 5–10 minutes and supported by a PowerPoint with approximately two slides. These meetings were an opportunity not only to clarify questions about the project and its implementation but also to receive feedback from the entire team regarding the current state of the project and the features to be integrated in the future. This is also why the project requirements evolved so significantly: although the main objective of this internship was clear (to create the visualization tool), both the structure of the interface and the remaining functionalities it should include were not clearly established, and the meetings were crucial to decide which ideas were appropriate, considering the internship's goals and duration.

Additionally, the project was developed within the LARA and Clava GitHub [15] repositories. However, instead of developing parts of the project and including them in the codebase iteratively, I was asked to create the solution in its own branches and only make it ready for integration at the end of the internship.

### 2.2 Stakeholders, Roles and Responsibilities

#### U.Porto Tutor:

- **Prof. Nuno Macedo:** Although he did not participate in the development of the project, he helped by answering technical questions about the elaboration of the deliverables for the Capstone Project, like this report.

#### INESC TEC Tutors:

- **Prof. João Bispo:** As the coordinator of the SPeCS team, he provided substantial support throughout the internship, from Clava's setup and usage to the troubleshooting of issues with the project and Clava itself. He also actively offered his insights on the project, contributing with several ideas for the functionalities and providing feedback on my implementation and design solutions. Additionally, he contributed with corrections and ideas for this report.
- **Prof. Luís Sousa:** As the most involved member in the development of LARA-JS and Clava-JS, which were the focus of this internship, he provided significant support in understanding the internship objectives and addressing questions about the project's architecture, features' implementation, and much more. He also contributed with multiple ideas for the interface and API.





As we can see, by the end of the two weeks, the project already met almost all the initial requirements. Consequently, it was decided in the meeting that, in the following weeks, I should focus on redesigning the interface and implementing additional features. Several ideas were discussed, including presenting additional information for each node in the interface, applying focus (i.e. scrolling the elements into view) to nodes and code when highlighted, and allowing users to provide only a subtree of the AST to the tool on request.

Moreover, during this phase, some issues were identified in the code of some types of nodes that differentiated them from the corresponding portions in the global code, hindering their identification. Therefore, I also committed to investigating these cases and applying corrections as needed.

### 2.3.2 Second Weekly Meeting

In the third week, I primarily focused on the **interface redesign**. To do this, I decided to create a mockup in Figma [10], with the look and functionalities that I intended for the project, at the end of the internship. Figma allows the creation of mockups that closely resemble a real application while enabling simpler and more intuitive style editing than by writing HTML and CSS code. This let me present my redesign ideas to my supervisors quicker and, consequently, receive their feedback faster.

Once the mockup was completed and approved, I proceeded with the refactoring of the CSS, to provide the tool with its new appearance. I also took the opportunity to integrate **extra features**, such as allowing the descendants of an AST node to be collapsed.

Furthermore, I worked on a small improvement in the communication with Clava via WebSockets, to always attempt the re-establishment of the connection with the compiler once interrupted or closed.

At the end of this meeting, I affirmed that I would continue working on the remaining functionalities, as well as investigating the feasibility of integrating syntax highlighting, as I had already attempted and faced several obstacles. It was also during this meeting that I proposed the creation of a system/framework that abstracts operations dependent on the compiler or language to the respective compiler. It is important to note that, following Prof. Luís Sousa's suggestion, and diverging from the initial goal of integrating the tool only into Clava, I decided to integrate the tool directly into the LARA framework, so that it could be reused for other LARA-based compilers. Nonetheless, the integration with Clava began to present situations where it was necessary to apply certain corrections that were only meaningful for C/C++ code and the Clava AST. This system would be a solution to make the tool not bound to a particular compiler.

### 2.3.3 Third Weekly Meeting

With this meeting, I marked the completion of the **compiler-abstracted system** I suggested last week, along with the application of more code fixes. It was also during this week that I finished almost all the planned functionalities, except for the division of the source code shown in the interface into its files and the syntax highlighting.

Consequently, it was decided in the meeting to finish the remaining functionalities, to test the tool more extensively, to find more bugs and necessary code fixes and to perform a final code refactoring, along with its documentation.

In addition, the other interns and I took the opportunity to clarify some questions regarding the documents we had to submit at the end of the current week (report and presentation support).

### 2.3.4 Report and Presentation Support Delivery

To conclude the internship, each intern was asked to prepare a brief one-page **final report**, summarizing the internship and its results, as well as the support for the **final presentation** in the final session.

### 2.3.5 Fourth Weekly Meeting

This was the last meeting dedicated to the internship. In it, I presented the final state of the project, with the planned functionalities fully implemented, the final **code refactoring** completed and **code documentation** done, being ready to be added to the LARA and Clava repositories.

Nevertheless, this meeting took place on the day before the final session, and its main goal was to rehearse the interns' presentations to the supervisors, so they could provide feedback and suggest corrections and improvements.

### 2.3.6 Final Session

This session marked the conclusion of the INESC TEC 2024 Summer Internship program. The purpose of this session was to celebrate the end of the internships, with each intern performing a **final presentation** of their internship to the other participants and a board of judges.

Each presentation had a minimum duration of 4 minutes (or 6 minutes for groups of two or more people), and the intern had to provide a brief introduction of themselves and their internship, discuss the process, mention the results, and conclude with what they learned from the internship and their experience. This presentation would then be used as the main evaluation element for the best internship award.

### 2.3.7 Gantt Chart

The Figure 2 shows a Gantt chart that summarizes the internship's schedule.

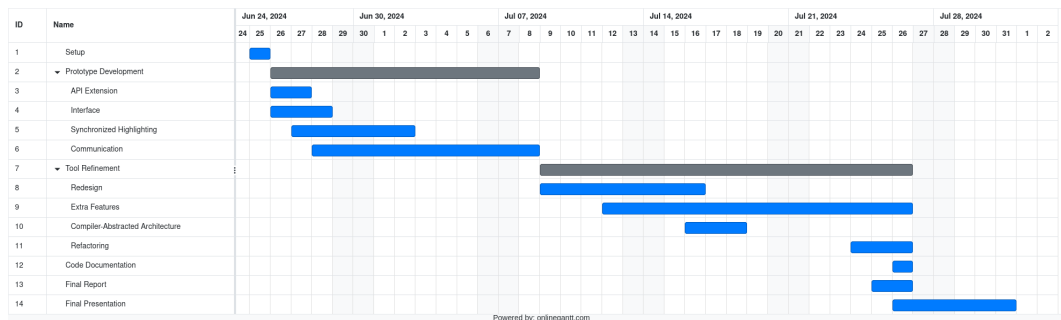


Figure 2: Gantt chart

## 3 Solution Development

### 3.1 Requirements

Starting this internship, I was not provided with a clear list of the requirements the final solution should meet. Nevertheless, from the initial internship proposal, a list of functional and non-functional requirements can be extracted, which I present in this section.

#### 3.1.1 Functional Requirements

- **Side-by-side visualization:** The web interface must allow the simultaneous visualization of the AST and the respective source code.
- **Synchronized highlighting between AST and source code:** When the user hovers the mouse over a node in the AST displayed on the web interface, the corresponding code segment should be highlighted, and vice-versa.
- **API Integration:** The tool should provide adequate functions in the API to execute and control it from the script given to Clava.
- **Communication with Clava:** The tool must be able to communicate with Clava, for updating and further control over the transpilation process.

#### 3.1.2 Non-Functional Requirements

- **Efficiency:** The project should use as less computer resources as practical, and it shouldn't add any undesired interference to the processing done by Clava.
- **Performance:** The tool must be able to parse the AST and code and perform the visualization efficiently, with minimal delays and response times, to ensure a smooth user experience. This also applies to the communication with Clava.
- **Usability:** The interface should be intuitive and easy to use, especially for new users who aren't familiar with the tool or Clava.
- **Compatibility:** The tool must work in all major operating systems, with no system-dependent problems, and the interface should be compatible with all modern browsers.
- **Maintainability:** The project's code should be well-structured and documented, following Clava's and LARA's code styles, logic and architecture.

### 3.2 Architecture and Technologies

#### 3.2.1 Architecture

From an architectural point of view, the tool can be divided into two parts, the server (or backend) and the user interface (or frontend). The **server** is responsible for processing and providing the necessary information to the interface (using the AST). While the **web interface** performs the visualization of the AST to the user, with the highlighting and other visualization functionalities. Some communication is also performed between the two for execution control. This architecture is illustrated in figure 3.

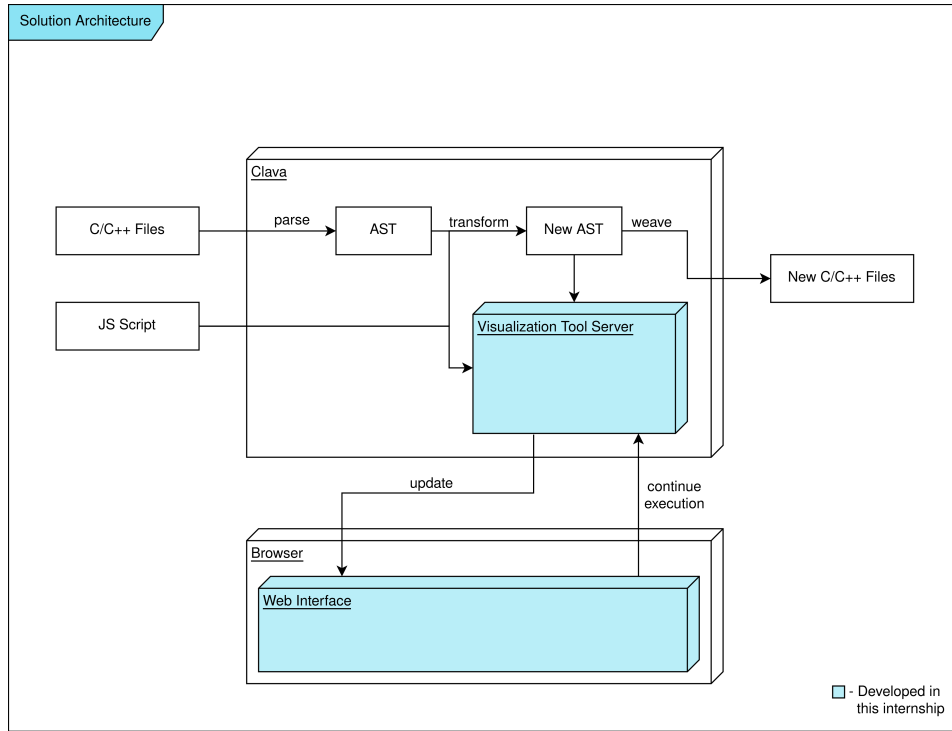


Figure 3: Solution architecture

### 3.2.2 Clava/Backend Technologies

#### JavaScript

JavaScript is a lightweight, interpreted programming language, that exhibits many particular characteristics, like being prototype-based, multi-paradigm and single-threaded. It is mostly known for its use as a scripting language for web pages, but it's also used on other non-browser environments, such as Node.js [7].

In the Clava project (and other LARA-based compilers), JavaScript is the base language for the LARA scripts that are given to the compiler. A LARA script is just a JavaScript script that uses the LARA API [24], allowing them to control Clava. Also, there has been an effort to migrate parts of both Clava and the LARA framework, which are written in Java, to JavaScript, to increase compatibility. These rewrites are named Clava-JS [23] and LARA-JS [25], respectively, and they were the versions of the projects this internship focused on. Although the source code of both projects is written in TypeScript, the exported APIs are in JavaScript.

#### TypeScript

TypeScript is a strongly typed programming language, developed by Microsoft, that can compile to JavaScript. It's a superset of JavaScript, using its syntax, but its core advantage is the addition of static types and type-checking [29], which is very useful to ensure safe code for large projects. This was the language that I predominantly used in this project: since Clava-JS and LARA-JS are written in TypeScript, every inclusion I've done to their APIs was written in the language.

## **Node.js**

Node.js is a runtime environment that allows JavaScript to be executed outside the browser. Built on Chrome's V8 JavaScript engine, Node.js is known for its event-driven, non-blocking I/O model [17], making it ideal for building scalable and high-performance web applications. Node.js also has a default package manager, called the Node Package Manager (or npm), which allows developers to install packages, manage dependencies and execute tasks with simple commands [16]. Clava-JS and LARA-JS are built on a Node.js environment, which allowed me to use the core libraries of Node, like the http package, for launching the web server. It also let me easily include other needed packages in the project through npm, like Express.js.

## **Express.js**

Express.js is a minimal and flexible Node.js web application framework, typically used for backend solutions. It includes an API, with HTTP utility methods, middleware integration and other features typically needed for web applications, with minimal performance drawbacks [12], simplifying the process of building server-side applications. Therefore, I used Express.js middleware to set the URL paths of the files needed for the web page, making it much easier to have the web interface up and running.

## **WebSocket API**

WebSocket is a computer communication protocol that provides bidirectional communication between client and server. WebSocket allows contiguous exchange once the connection is established, removing the need for reconnection for each communication, making it ideal for performing real-time communication [14]. To use the WebSocket communication on a web page, one can use the WebSocket API, which is available in almost all browsers [8]. This is the reason I used it in this project, since I needed a way to quickly transmit and receive data from the tool server.

Yet, the WebSocket API did lack reconnection capabilities, i.e. when the Clava execution ends, the server will be closed, the client will be disconnected and it will not reconnect automatically. This can be solved by polling the connection when disconnected. This does bring some delays in the reconnection, caused by the browser, but, from my experimentation, it does not appear to be a significant issue for normal usage.

## **Clang**

The Clang compiler is an open-source compiler for the C family of programming languages. It builds on the LLVM optimizer and code generator, allowing it to provide high-quality optimization and code generation support for multiple targets [26]. This is the main compiler used by Clava and, although this internship did not work with Clang directly, some problems faced and design choices made were caused by its particularities.

### 3.2.3 Frontend Technologies

#### HTML

HTML (HyperText Markup Language) is the standard markup language for specifying the structure of web pages. It is, therefore, an essential component of web development. HTML documents are composed of elements, which are surrounded by tags [6]. There are many Node JavaScript frontend frameworks that prevent the programmer from handling the page HTML directly, but using pure HTML code helped to keep the interface as lightweight as possible.

#### CSS

CSS (Cascading Style Sheets) is the standard language for controlling the style of a web page [5]. It allowed me to apply the style that can be seen in each version of the web interface.

#### TypeScript

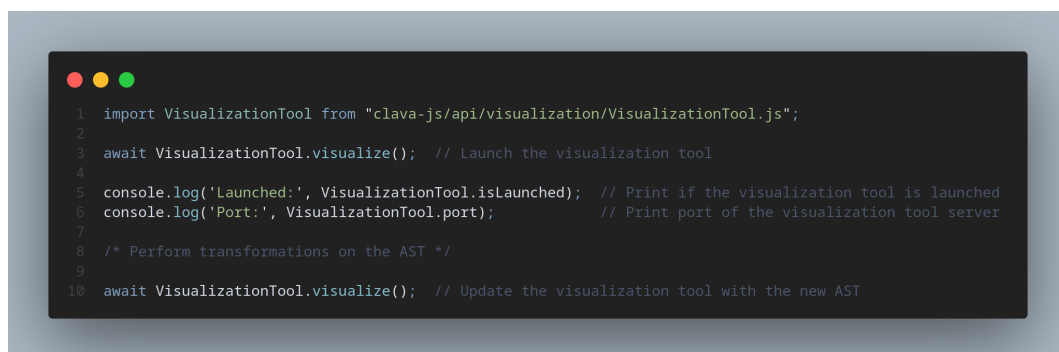
TypeScript was also used as the main scripting language of the web interface, allowing for complex frontend functionality, like custom element behavior and connection with Clava.

## 3.3 Developed Solution

### 3.3.1 API Extension

Clava is a source-to-source compiler that applies transformations to the source code, based on the given script. This script must have access to the Clava and/or LARA APIs, which contain the methods needed to manipulate the AST and further control the compiler. For the user to be able to launch the visualization tool, it is necessary to extend the API with adequate and well-documented functions for managing the tool.

The tool methods that were integrated into the API can be found in the declaration of the `GenericVisualizationTool` class, found in the LARA API. A basic usage example can be found in the Figure 4.



```
1 import VisualizationTool from "clava-js/api/visualization/VisualizationTool.js";
2
3 await VisualizationTool.visualize(); // Launch the visualization tool
4
5 console.log('Launched:', VisualizationTool.isLaunched); // Print if the visualization tool is launched
6 console.log('Port:', VisualizationTool.port);           // Print port of the visualization tool server
7
8 /* Perform transformations on the AST */
9
10 await VisualizationTool.visualize(); // Update the visualization tool with the new AST
```

Figure 4: Basic usage example of the API

The `visualize` method will cover most use cases. This method, when executed, will launch the tool server, if already not launched, update the tool with the new code and pause the execution of the Clava script, waiting for the response of the web interface to continue. It will also automatically output the server URL to the console, to be opened in the browser.

So, when the user wants to visualize the current AST, it must make a call to `visualize`. The call must be made with the `await` keyword, since JavaScript is single-threaded [7]: if we try to run without waiting for the promise, as soon as the execution of the asynchronous function is interrupted, the execution of the script will be resumed, eventually finishing and closing the web server, before the user can even open the interface. The keyword ensures that the compilation is resumed only when the asynchronous function halts, i.e. when the response from the interface is received.

The remaining properties of the API extension are just for reading the tool server info, like if the tool has been launched and to what URL the server is listening.

### 3.3.2 AST and Source Code Visualization

As one of the main objectives of the internship, the interface is able to display the AST and the corresponding code, side-by-side. The final design of the interface is found in Figure 5, which took the Flutter Inspector [11] interface as the main source of inspiration. After waiting to receive the AST and source code from the server, the tool presents them both to the user. They are the main elements of the interface, appearing in two adjacent containers.

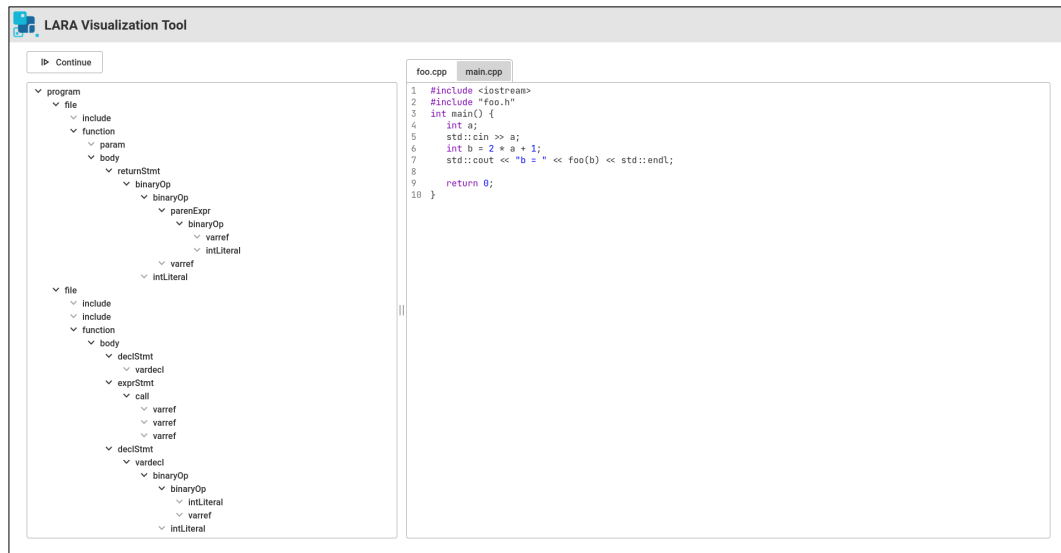


Figure 5: Final interface design

The nodes of the AST are shown in a hierarchical list. For each non-leaf node, it is possible to collapse/expand its descendants by left-clicking the chevron to its left. This helps reduce the length of the AST displayed, since it tends to get pretty extensive for larger programs.

The code also appears divided into files and just one is presented at one time, which is also useful for when the code spans multiple files. Another file can be selected from the file tabs, on top of the code container. In addition to that, to the left side, we have the line numbers, to help navigate through the code.

Furthermore, the interface was done ensuring that it follows good web standards, namely that the HTML and CSS code are valid and that it has no compatibility or accessibility issues.

### 3.3.3 Mapping Between AST and Source Code

To accomplish the task of mapping each AST node to the corresponding code and vice-versa, I added a highlighting functionality. This way, when an AST node is hovered by the mouse, that node and its corresponding code get highlighted in a light blue. The ancestors of the node also get highlighted with a lighter color, which helps understand the hierarchy of the code. Likewise, by hovering over a specific piece of code, the interface will highlight the deepest node that contains that code. So, if a return statement inside a function is hovered, the return statement's node will be highlighted rather than the function's or its body's.

To know which node and code are associated, the tool surrounds each one in a **span** HTML element and sets the node ID as an attribute, making the search trivial using an HTML query.

Moreover, a stronger highlight can be achieved by left-clicking the node/code, as we observe in Figure 6. This highlight will stay, even if the mouse is moved, and only one node/code block can be selected at a time, so clicking on another node will deselect the previous one. By highlighting the node this way, the user can also see some useful information about the node, in the bottom-right corner. This information box will also point out when the code of the node could not be found inside the program's source code, or when the node does not appear to have code at all.

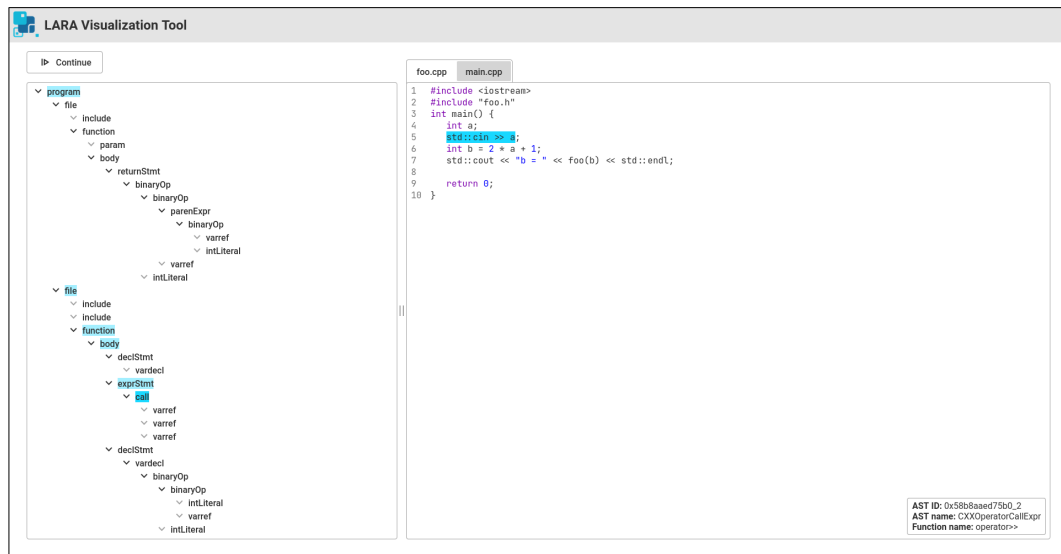


Figure 6: Highlight with mouse click

### 3.3.4 Script Execution Control

As already mentioned, the tool can control the execution flow of the script processed by Clava.

When the user calls the **visualize** method of the visualization tool from the API, the compilation is postponed. Aside from the reasons already mentioned, this is also to ensure the user has time to analyze the AST in intermediary steps of the transpilation. To continue the execution, the user must click on the “Continue” button on the tool’s web page. This button will normally be disabled unless Clava is waiting for a response from it



### 3.3.5 Syntax Highlighting

To ease the interpretation of the source code, as most code editors do, I added syntax highlighting to the code presented in the interface.

This was the extra feature that my tutors and I most debated about its feasibility. There are already a lot of solutions that can perform syntax highlighting on code of most languages, like Google's code prettifier [1] and highlight.js [13]. However, since I need to receive the source code with HTML tags inside it (for linking the nodes and the code), most of these scripts failed to properly highlight the code. The only exception was Google's code prettifier, which not only is an unmaintained project but also did fail in most of the cases, only working on some of them. Because of this, if the project is going to have syntax highlighting, it has to be done from scratch.

To accomplish this, before sending the source code to the browser, the tool identifies relevant words and expressions, frequently using regular expressions and the respective AST node type, and wraps them in HTML `span` elements, with the appropriate category. Although this identification can seem trivial, due to the categorization of nodes in the AST, it is not. For example, none of the keywords and types are isolated in their own AST node, so they must be searched inside the code of potential nodes. Currently, the only language constructs that are highlighted are literals (which are categorized between string and non-string literals), comments, types and keywords, which are almost the same as Google's code prettifier. The syntax highlighting is relatively basic and more adjustments and enhancements can be made, but, for the scope of this project, it was more than enough.

### 3.3.6 Other Interface Features

This section covers the features that I deemed not worth an isolated section on their own.

There can be situations where the horizontal space of the AST and code containers are a limitation, when the AST has a deep hierarchy (e.g. when the code is deeply nested) or the source code has long lines. To help with this, I implemented a resizer, which is found between the two main containers, that allows changing the containers' widths, while still fitting inside the viewport.

In addition, the interface having a dark mode can be useful for some users, since it causes reduced eye strain and improved readability in low light. So, a dark color scheme was created for the interface, which is automatically used if the browser has the dark theme as its default, in the browser settings.

### 3.3.7 Integration with Other Compilers

The initial internship proposal stated that the visualization tool should be made for Clava, without mentioning other compilers. Yet, Prof. Luís Sousa suggested building the tool inside the LARA framework since, on paper, the tool and the mapping performed should work for any source-to-source compiler based on LARA.

Even though the project started being developed on top of the LARA codebase, I quickly faced multiple issues. Firstly, the AST node's code is not directly accessible from the LARA's interface for AST nodes (the `Joinpoint` class), although this is planned to be resolved in the future. Secondly, for the linking between Clava's AST nodes and code, because of the way Clava provides the code of each node, and some transformations it performs internally, the tool would fail to find the node code in the global code and, therefore, the visual mapping. Let's illustrate this with the following example, represented in Figure 7: suppose we have two integer variables declared in the same line.

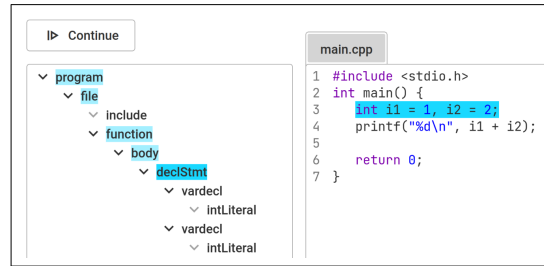


Figure 7: Declaration of two variables in a single line

This portion of the code is translated into a `declStmt` node (declaration statement) with two `vardecl` children (variable declaration). A particularity of Clava is that it will consider the code of each variable declaration as a declaration in its own line. So, the tool will be able to match the code of the first declaration, as it is `int i1 = 1`, but it will fail for the second declaration, since the associated code is `int i2 = 2`, which is not part of the code of the parent node.

Situations like this forced me to apply corrections to the code of certain nodes. In this example, the type of the second child's code must be removed. The problem is that these corrections are only meaningful to Clava, and can break the tool for other compilers. Also, there are other compiler/language-dependent operations, like the syntax highlighting and retrieval of additional node information. To solve this major complication, I structured the tool code in a system that abstracts all these operations to the compiler. The structure developed is illustrated in the class diagram in Figure 8.

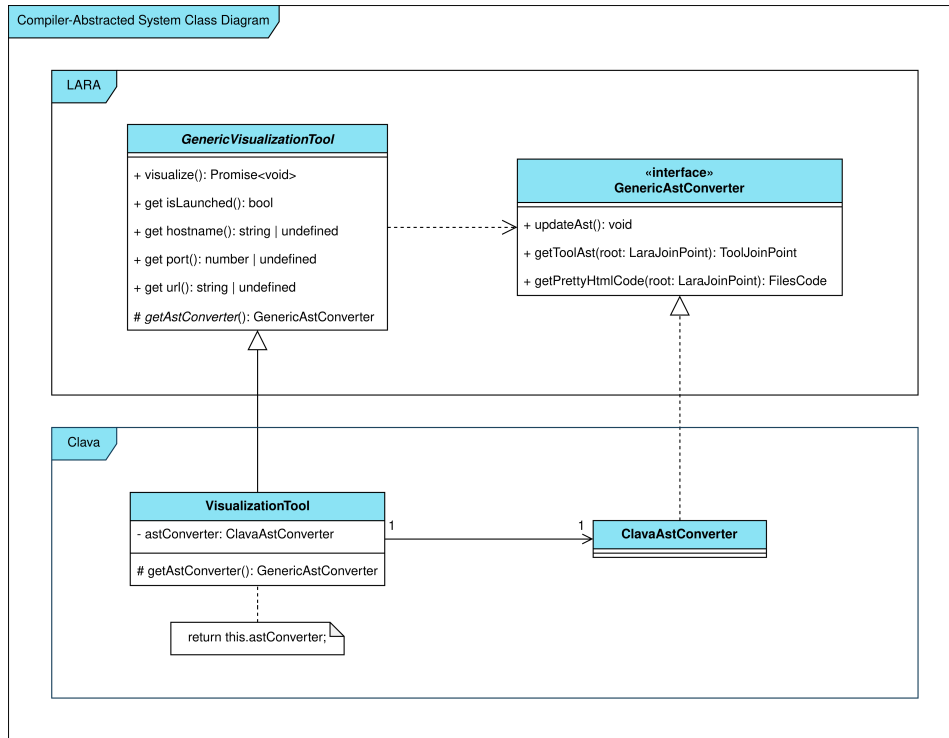


Figure 8: Compiler-abstracted system class diagram

This is an implementation of the Factory Method design pattern [18]. The frontend files are included in the LARA framework codebase, and the `GenericVisualizationTool` abstract class carries out the operations referred for the API, i.e. launching the web server and communicating with the browser. To include the tool in some compiler, it must create a class that implements the interface `GenericAstConverter`. This interface includes only three functions:

- `updateAst`, that rebuilds the AST in the compiler (this is useful for maintaining attribute consistency, such as location references).
- `getToolAst`, which returns the AST in a representation that the tool can use, with just the information needed from each node.
- `getPrettyHtmlCode`, which returns the source code of each file, as a string, ready to be inserted in the web page document, with all the needed HTML tags for the node-code linking and, perhaps, syntax highlighting already inserted.

Then, it must create a class that inherits from `GenericVisualizationTool` and override its `getAstConverter` method to return an instance of the created implementation of `GenericAstConverter`. From there, the API methods should be accessed from an instance of the `GenericVisualizationTool` derived class. In the case of Clava, this is the `VisualizationTool` class.

By following this code structure, we perform the least number of operations possible in the compiler, maintaining the tool core implementation generalized. This way, the visualization tool is not bound to any particular compiler or language and can be easily included in any LARA-based compiler.

### 3.4 Validation

The validation of the developed solution was done mainly through two methods.

To verify that the tool's behavior was working as intended, especially the integration with Clava, I conducted extensive stress testing throughout the internship. For this step, it was important to use diverse source code examples, especially with language features that are known for causing the most problems for compilers. Therefore, Prof. João Bispo suggested that I use the source code used by CACTI, a compiler analysis tool, developed for another Capstone Project in the SPeCS group, that tests, studies and compares the capabilities of different compilers, using C and C++ files with features that are often problematic [22]. With these tests, along with code that I created and searched for, I was able to identify a lot of the compatibility issues with Clava, which I eventually fixed, as well as other errors and bugs, e.g. in the communication between the server and the browser.

In addition, I received constant feedback from my tutors about the implemented functionalities and the interface's overall look and feel. In every weekly meeting, and even outside the meetings, we discussed considerably about the developed features, what should be implemented, what should not, and what should be changed, always with the end user preferences and the internship scope in mind. When I finished a feature, or when I created the new design, they were the first to know and to give their opinion. I also sought the opinions of other members of the SPeCS group, who contributed with their own ideas and opinions as well, due to their experience using Clava.

## 4 Conclusions

### 4.1 Achieved Results

In the end, I can confidently affirm that this internship yielded very impressive results. Not only I was able to achieve the desired goals, but I surpassed them, by delivering a very complete and easy-to-use visualization tool, with many extra functionalities, and that can be extended to work with other compilers, not just Clava. Moreover, all the non-functional requirements were met, as the features that are included proved to be working as intended in all tested scenarios, the tool shows to be efficient and easy-to-use and the code follows the codebases' standards and good web practices. This all reflected on my supervisor's excellent appraisal in the internship's final report.

Additionally, I was very excited about the purpose of this project. It solves a problem that all users of Clava face (lack of visualization capabilities), and I already received a lot of support and excitement from some users over the results of this internship. It means the world to me to be able to contribute this much to a bigger organization with something I poured my heart into.

Aside from the project, I am also proud of my integration into the SPeCS team. From the start, I managed to build good relationships with most of the members, and I also managed to communicate and work well with the team. The work environment in the SPeCS room is very supportive, inclusive and engaging, without being professional. Although I consider I was very autonomous during the development process, I could always count on my tutors and the other members for help.

### 4.2 Lessons Learned

With this internship, I had the opportunity to gain more experience as a software developer. Not only I was able to hone my skills in web development, deepening the knowledge I acquired while taking the "Web Languages and Technologies" curricular unit, as well as other competencies I learned in other curricular units, but I also could perfect my abilities in working on a relatively large-scale project, communicating with multiple team members, focusing on code clarity and consistency and documenting my progress along the way. Furthermore, since I worked every day with Clava, even though my project didn't engage with the compilation process directly, I could get a better grasp of the working mechanisms of compilers, which I didn't quite have at this point in my major.

Moreover, with this internship, I was able to work on other soft skills. The one I consider I worked on the most was user empathy: developing a project with relatively loose requirements had the adversity of having to find and choose what features would benefit the end users the most, which is very important for almost all software projects. Nevertheless, some features implemented were a challenge and required a lot of my problem-solving and creative thinking skills and, as always, team communication was important.

Furthermore, this internship was an opportunity to get closer to the technology research reality, which was one of my main objectives. In the SPeCS room, I could see first-hand the daily life of researchers, the way they organize meetings and their topics of research, which obviously fed my interest in the area. I even got to see and support the rehearsals of the presentations of some Masters students' theses.

### 4.3 Future Work

The final solution did meet all the requirements initially set, and many additional functionalities were added, with everything working as expected, so there isn't much that should be improved.

Nonetheless, one thing that I wanted to try, before the end of the internship, was to test the Socket.IO API for the communication with the server instead of the WebSocket API. Socket.IO is a library for low-latency, bidirectional and event-based communication, much like WebSocket. It even uses WebSocket under the hood [20]. However, a client using Socket.IO will automatically try to reconnect after a small delay [19]. This could be a solution to the reconnection delay problem that the WebSocket API presents, even though it is not a major setback. Also, some parts of the code can be improved as the Clava project evolves, like simplifying some logic against Clava bugs, once they are fixed.

In relation to the development of new features, there isn't much to suggest. One obvious idea is the integration of the tool into other existent compilers (which should be relatively easy with the tool's generalized architecture). Another idea I had is the inclusion of Clava operations to the visualization tool, e.g. searching an AST node using the API's query operations or modifying the AST nodes directly in the interface.

## References

- [1] Google Archive. JavaScript code prettifier. <https://github.com/googlearchive/code-prettify>, 2020. [Online; accessed 9 August 2024].
- [2] Keith D. Cooper and Linda Torczon. Chapter 1 - overview of compilation. In Keith D. Cooper and Linda Torczon, editors, *Engineering a Compiler (Third Edition)*, pages 1–26. Morgan Kaufmann, Philadelphia, third edition edition, 2023.
- [3] Keith D. Cooper and Linda Torczon. Chapter 5 - syntax-driven translation. In Keith D. Cooper and Linda Torczon, editors, *Engineering a Compiler (Third Edition)*, pages 209–273. Morgan Kaufmann, Philadelphia, third edition edition, 2023.
- [4] MDN Web Docs. Server. <https://developer.mozilla.org/en-US/docs/Glossary/Server>, 2023. [Online; accessed 12 August 2024].
- [5] MDN Web Docs. CSS. <https://developer.mozilla.org/en-US/docs/Glossary/CSS>, 2024. [Online; accessed 15 August 2024].
- [6] MDN Web Docs. HTML. <https://developer.mozilla.org/en-US/docs/Glossary/HTML>, 2024. [Online; accessed 15 August 2024].
- [7] MDN Web Docs. JavaScript. <https://developer.mozilla.org/en-US/docs/Web/JavaScript>, 2024. [Online; accessed 9 August 2024].
- [8] MDN Web Docs. The WebSocket API. [https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\\_API](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API), 2024. [Online; accessed 9 August 2024].
- [9] MDN Web Docs. URL. <https://developer.mozilla.org/en-US/docs/Glossary/URL>, 2024. [Online; accessed 12 August 2024].
- [10] Figma. Figma: The Collaborative Interface Design Tool. <https://www.figma.com/>. [Online; accessed 9 August 2024].
- [11] Flutter. Use the Flutter inspector. <https://docs.flutter.dev/tools/devtools/inspector>, 2024. [Online; accessed 9 August 2024].
- [12] OpenJS Foundation. Express - Node.js web application framework. <https://expressjs.com/>. [Online; accessed 9 August 2024].
- [13] highlight.js. highlight.js. <https://highlightjs.org>. [Online; accessed 9 August 2024].
- [14] IBM. What is an API (Application Programming Interface)? <https://www.ibm.com/topics/api>, 2024. [Online; accessed 12 August 2024].
- [15] GitHub Inc. GitHub. <https://github.com/>. [Online; accessed 9 August 2024].
- [16] Node.js. An introduction to the npm package manager. <https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager#introduction-to-npm>. [Online; accessed 11 August 2024].
- [17] Node.js. Introduction to Node.js. <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs>. [Online; accessed 11 August 2024].
- [18] Refactoring.Guru. Factory Method. <https://refactoring.guru/design-patterns/factory-method>, 2024. [Online; accessed 9 August 2024].

- [19] Socket.IO. Handling disconnections. <https://socket.io/docs/v4/tutorial/handling-disconnections>, 2024. [Online; accessed 9 August 2024].
- [20] Socket.IO. Introduction. <https://socket.io/docs/v4/>, 2024. [Online; accessed 9 August 2024].
- [21] SPeCS. LARA Framework. <https://github.com/specs-feup/lara-framework>, 2019. [Online; accessed 10 August 2024].
- [22] SPeCS. CACTI. <https://github.com/specs-feup/cacti>, 2023. [Online; accessed 9 August 2024].
- [23] SPeCS. Clava-JS. <https://github.com/specs-feup/clava/tree/master/Clava-JS>, 2023. [Online; accessed 11 August 2024].
- [24] SPeCS. Clava Wiki. <https://github.com/specs-feup/clava/wiki>, 2023. [Online; accessed 10 August 2024].
- [25] SPeCS. LARA-JS. <https://github.com/specs-feup/lara-framework/tree/master/Lara-JS>, 2023. [Online; accessed 11 August 2024].
- [26] The Clang Team. Clang Compiler User’s Manual. <https://clang.llvm.org/docs/UsersManual.html>. [Online; accessed 9 August 2024].
- [27] INESC TEC. INESC TEC. <https://www.inesctec.pt/en>, 2024. [Online; accessed 9 August 2024].
- [28] Slack Technologies. Slack. <https://slack.com/>. [Online; accessed 11 August 2024].
- [29] TypeScript. TypeScript for the New Programmer. <https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html>. [Online; accessed 9 August 2024].