OML Alexandria

Team A1 - Pteam Ptolemy

Group Members:
Matthew Craig
Jeffrey Ma
Tamjid Khan
Jacob Linder
Jacques Gueye

Github: https://github.com/0x65-e/oml-alexandria

Motivation/Concept:

Systems Engineering is an important subfield of engineering that focuses on integrating complex systems together. An important aspect of any engineering practice, especially systems engineering, is problem definition and analysis. OML (Ontological Modeling Language) is a language that defines systems engineering vocabularies and uses them to describe systems, which is critical for capturing knowledge. To enable widespread support of OML and streamline the process of writing, validating, and tracking OML files, it is critical to have language support tooling integrated into popular development platforms.

Microsoft Visual Studio Code is one such popular source-code editor for development in many languages. VS Code is one of the most popular editors in the world. It is cross-platform and even supports browser-based instances like vscode.dev, gitpod and codespaces. Among the benefits of cloud-based editors are that they require no installation, update automatically, can easily sync across devices, perform better on low-powered hardware, standardize the editing experience across heterogeneous devices, and can be accessed from almost anywhere.

One of the reasons for VS Code's widespread popularity is that it can support development in nearly any language through the use of extensions that implement the Language Server Protocol (LSP). LSP defines an application-neutral interface for editors like VS Code to communicate with a language server that provides helpful features like auto complete, go to definition, and code linting. Implementing such a language server used to be a tedious task. However, new frameworks like Langium have made this significantly easier. Langium can generate a language server with many features implemented by default for any language defined with a BNF-like grammer (including OML). Since Langium is implemented in TypeScript, the server can be easily wrapped in a VS Code extension and run on the client instead of requiring a separate server.

OML currently has an Eclipse-based editor called Rosetta and a VS Code extension called Luxor. Both of these development environments require the OML language server to run as a separate process on the client machine or on a separate server, which makes using browser-based editors difficult or impossible for OML development. Our project is to develop a proof of concept cloud-friendly OML development extension for VS Code that works with web-based editors natively. It will implement common language server functionality through the use of Langium. We also intend to implement logical validation for OML files (beyond syntax validation). If time permits, we will strive for feature parity with the existing editor implementations.

One of the primary features of the extension is to enable users to visualize their OML descriptions using diagrams. This is a critical feature of the existing OML Luxor extension, and we intend to replicate this functionality.

Links to Epics:

https://github.com/0x65-e/oml-alexandria/issues

Technology Stack:

We have chosen Langium as the framework for our language server implementation, Sprotty for the front-end visualization framework, and Node.js as the runtime environment. By implementing our extension almost exclusively in TypeScript, we aim to create a serverless solution that can be run in the browser. Our choice of Node.js is driven by VS Code, which will allow us to run our extension in browser-based editors.

Langium is an open source language engineering tool with first-class support for the Language Server Protocol, written in TypeScript and running in Node.js. Langium will allow us to implement a language server for VS Code with the possibility to extend to other supported platforms in the future, like Eclipse Theia. Essentially, Langium enables us to quickly build a backend with basic language support simply by providing OML's BNF grammar. We can later extend Langium's default implementation with additional features.

Sprotty is a web-based framework for diagrams. Since Sprotty and Langium are both implemented in Typescript and running on Node.js, we believe that it should be straightforward to integrate Sprotty as the visualization library for our extension. Sprotty will require both a backend extension of the language server (to send diagram data to the frontend client), as well as frontend code to interpret and display the data in the proper diagram format.

We aim to organize our project in the following way:

- Extension
    - Entry point for the vs-code extension
        - Contains all VS Code specific functionality, including launching commands and views
    - Client for LSP and Sprotty
- Language Server
    - Langium OML implementation for basic features
    - Additional semantic validation for OML
- Webview
    - Visualization of OML implementation (using Sprotty)

We believe that these frameworks and tools will enable us to complete our goals within the project timeline. We may add additional tools as necessary.

Feasibility:

Before beginning development of a large-scale project such as this, we must first analyze the feasibility of the project. This process is to determine whether our team has the time, skills, motivation, and resources to complete the entire proposed project before the quarter is over.

In order to build an OML Langium extension, our whole team must first have an understanding of the overall goal of the project. Given that each of us has significant familiarity with writing software using programming languages within various IDEs, and that all of us not only have beforehand used, but even continue to use language extensions within our IDEs that increase our efficiency in software development, the aforementioned extension project is ultimately one that lands close to home for all of us as individuals. Thus we may achieve intimate familiarity with the function of such an extension, not only as to understand the set of requirements soon to be documented in regards to their practical programmatic implementation, but in that we experientially know the utility and nuances that these extensions bring to the software development cycle. Lo! As it is said, they are to our daily coding sessions as a fork and knife are to the evening meal.

Having demonstrated our understanding of the task at hand, we must assess whether our team possesses the skills and knowledge to partake in the development of this project. Much of our development will be in TypeScript. One challenge is that some of our team has never used TypeScript, and others only have limited experience. However, each of us has at least some familiarity with development in JavaScript, so we expect to be able to pick up TypeScript quickly, especially with the wide variety of resources available on the web. We also have done a lot of programming in other object oriented languages, particularly through the many courses at UCLA that focus on C++. We each have picked up many different programming languages through various classes and jobs, so learning TypeScript should be no great difficulty, especially in considering the previous two points.

We also are able to understand the project architecture from a high level, since we have each built and used parsers and have familiarity with servers from CS classes. This gives us enough knowledge about the various applications and libraries we are using, so that we will know how to create a program that can treat each of their implementations as black boxes, making development on our end much simpler. The frameworks we have chosen are well suited to our project, and they will aid our development by solving some of the most difficult problems (language parsing, the mechanics of visualization), allowing us to focus on the application logic. Although we are not familiar with using OML, Sprotty, Langium, or the Language Server Protocol, which presents some challenge, we have used similar technologies and understand them to the degree that their respective documentations should be more than sufficient to provide understanding as to how we incorporate them in our implementation.

Although each of us are busy with school and extracurriculars, we have allotted enough time in planning our schedules that we should have plenty of time to finish the project, and possibly even to add or improve features of the extension. We will have to spend the first week of the project getting up to speed with the frameworks and languages we are using, but after this period, we are all capable software engineers with relevant experience and so development should not present too much difficulty.

Capability:

Front-end Development

Tamjid has experience working with mobile app development using React Native. As such, he is familiar with client-side development technologies such as HTML, CSS, and Javascript. He is also familiar with the testing and debugging client-side applications to optimize user experience. As the front-end development required for this project is not particularly difficult, this level of experience should be sufficient for the scope of the project.

Back-end Development

Matthew briefly worked on a VS Code language server extension at an internship, so he has a passing familiarity with language servers and with VS Code extensions. The language server was written in Rust with a Typescript wrapper instead of native Typescript, but he is confident that he can apply his previous experience with LSPs while learning Typescript. Matthew has experience with language syntax, parsing, and semantics from CS 131, and with knowledge representation and logical inference from CS 161. These classes will be useful implementing the back-end of the language server and logical validation for OML.

Jeffrey has experience with static analysis and compilers from both a previous internship as well as research. He has taken CS 131 and is equipped to work on the language server and validation. Jeffrey is confident in learning Typescript due to familiarity with Javascript. Jeffrey is also familiar with BNF grammars and languages from CS 181, as well as ASTs which will be helpful for understanding the OML syntax and working with Langium.

Jacob has limited experience with TypeScript through a UCLA campus job where he helped develop the USAC Expenditure Viewer in React. He has also taken CS 131, CS 181, and LING 20, which gave him a good understanding of grammatical structure and language parsing. He has experience with backend servers, mainly from working in GCP for personal projects, as well as server development in some smaller UCLA projects.

Jacques has some experience with TypeScript from using a query language called GraphQL, which lets him use TS to make requests. He has taken CS 181 and CS 131, which allows him to have a good understanding of BNF grammars. He will also help frontend, to which he is familiar with through his prior internship with a web app.