Introducing BIGUML: A Flexible Open-Source GLSP-based Web Modeling Tool for UML

Haydar Metin and Dominik Bork

Introducing BIGUML: A Flexible Open-Source GLSP-based Web Modeling Tool for UML

Haydar Metin
Business Informatics Group, TU Wien, Vienna, Austria
haydar.metin@tuwien.ac.at,

Dominik Bork
Business Informatics Group, TU Wien, Vienna, Austria
dominik.bork@tuwien.ac.at,

***Abstract*—Unified Modeling Language (UML) plays a crucial role in software development by providing a standardized notation for visualizing, specifying, and documenting system architectures. Traditional UML modeling tools often follow a rich client approach and suffer from limitations such as steep learning curves and restricted extensibility. Usually, those tools are also proprietary and constrained to specific platforms. This paper presents BIGUML, a flexible, open-source UML modeling tool implemented as a Visual Studio Code extension leveraging the Graphical Language Server Protocol (GLSP) to offer a seamless modeling experience with rich features and enhanced usability. The early release already supports the Class and the Use Case diagram, further UML diagrams will follow soon iteratively. We introduce BIGUML and highlight, how its flexible architecture allowed the integration of custom features, such as the property and outline view, and contextual copy-paste operations. The BIGUML modeling tool is one of the first tools to incorporate GLSP and to be distributed through the VS Code ecosystem.**

***Index Terms*—UML, Software modeling, GLSP, Modeling tool, Web modeling, LSP**

I. Introduction

Unified Modeling Language (UML) is a foundational notation for visualizing, specifying, constructing, and documenting complex software systems. UML diagrams provide a standardized way to communicate design concepts, architecture, and behavior among software development teams. The development of modeling tools has a long tradition within the MODELS community and the broader modeling communities [1]. For this reason, the modeling community has developed many UML tools in the past [2]. However, a recent survey and evaluation showed the current tools are not satisfactorily addressing the requirements of their users such as effectively assisting them on creating models or providing enough context information [3]. To address this issue, there is a need for modern, flexible, and open UML modeling tools that facilitate advanced visualization and interaction functionalities (cf. [4]). In response to this need, the Language Server Protocol (LSP) and the Graphical Language Server Platform (GLSP) have gained popularity for their capabilities in enhancing code editors' functionality and are used in numerous popular IDEs like Eclipse, Theia, and VS Code. A comprehensive discussion of the flexibility enabled by GLSP-based web modeling tools is provided in [5], [6].

Part of this research was funded through the FFG Innovationsscheck entitled 'Automatisiertes End-to-End-Testen von Cloud-basierten Modellierungswerkzeugen' (No. 903552).

In this paper, we present BIGUML. This UML modeling tool utilizes GLSP to enhance the UML modeling experience, which, to the best of our knowledge, is the first realized and released GLSP open-source UML editor for VS Code available. By utilizing the GLSP framework and a well-known domain like UML with multiple intricate complexities, we demonstrate that it is possible to have a paradigm shift from traditional modeling tools. We will provide an overview of BIGUML's theoretical aspects (Section II), followed by the features it exposes (Section III), a short discussion about available related tools (Section IV), and the future outlook of BIGUML (Section V).

II. The BIGUML Modeling Tool

The core technology utilized by BIGUML is the Graphical Language Server Platform (GLSP), which is an open-source framework designed for creating customized diagram editors using web technologies [6] hosted by the Eclipse Foundation. Editors developed using GLSP can be easily integrated into web applications and tool platforms like Eclipse Theia and VS Code, and even traditional Rich Client Application platforms like Eclipse RCP. Accordingly, GLSP adopts the structure and principles introduced by the Language Server Protocol (LSP) [7], [8]. However, GLSP extends LSP to address specific challenges that arise when dealing with graphical models as opposed to textual documents (cf. [9]). These challenges include transitioning from a two-dimensional representation (document row and character position) to a three-dimensional space (where elements occupy a geographical area and can include child elements) and moving from simple character-based editing operations to complex ones, such as creating relationships between nodes in a diagram and restricting allowed connections, among others [5], [6].

Fig. 1 showcases the various components within GLSP. Firstly, we have the source models, which contain the actual model data, such as a UML model. Next, we have the server framework, which holds language-specific knowledge and functionalities related to the model and domain. The third component is the GLSP-Client, a language-agnostic graphical modeling client. This client is responsible for rendering the diagram and managing user interfaces. Remarkably, the GLSP-Client can be seamlessly integrated into different platforms, including Theia and VS Code. Lastly, the GLSP-Client and server communicate with each other through messages defined

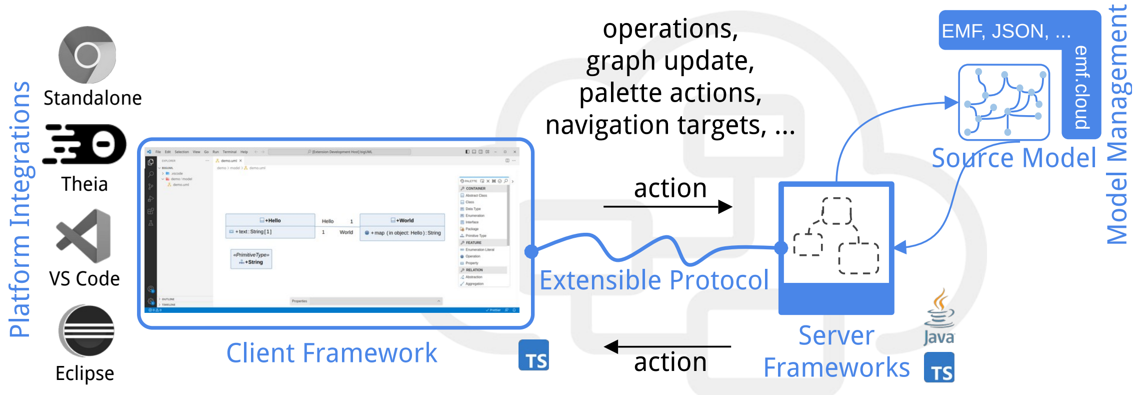

Fig. 1: Overview of GLSP components and their interplay [6]

in a flexible and extensible protocol. This protocol enables smooth and efficient data exchange between the client and server, allowing them to work together. More information regarding GLSP can be taken from their documentation[1] and from [6]. A discussion on how characteristics of graphical modeling languages can facilitate the flexibility of the core LSP protocol is presented in [9]. The subsequent sections focus on the GLSP-based architecture of BIGUML and the tool deployment within VS Code.

A. BIGUML *Architecture*

Fig. 2 illustrates the BIGUML reference architecture, which the different deployed components (e.g., servers), such as the GLSP-Server and the model server internally adhere to. For a detailed understanding of the reference architecture, the interested reader is referred to [5]. In this context, the BIGUML architecture aligns with the adaptable and reusable components presented there. In the following, we will provide only a brief overview of the fundamental principles that shape the BIGUML architecture.

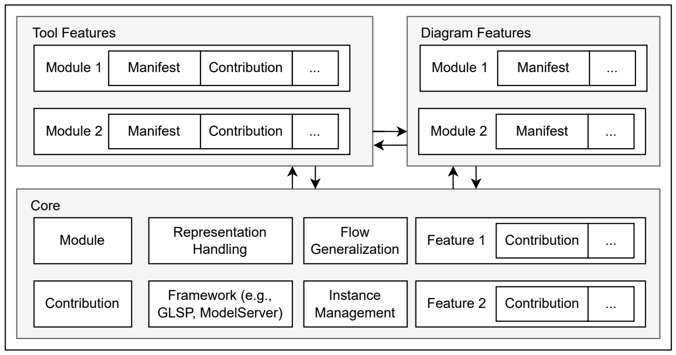

Fig. 2: BIGUML Architecture

BIGUML adopts a modular architecture. It leverages Dependency Injection and various design patterns such as Separation of Concerns, Single Source of Truth, and Single Responsibility Principle. Furthermore, the architecture is categorized into three main components: *Core*, *Tool*, and *Diagram features*, each responsible for governing different aspects of the application.

[1]https://eclipse.dev/glsp/documentation/overview/

- *Core features:* These components directly interact with the server and the underlying framework and serve as entry points for other features. They lack language-specific information while acting as a bridge between the server and other functionalities. Overall, they manage the server application and handle any changes introduced by the evolving framework.
- *Diagram features:* These components offer language-specific functionality and have access to the source model. They facilitate CRUD (Create, Read, Update, Delete) operations for other features to control the modifications of the source model.
- *Tool features:* These components extend the functionality by utilizing Core and Diagram features. They provide additional language-agnostic capabilities that are not present in the server framework, such as custom diagram outlines, copy-paste, and auto-complete.

Moreover, within each module, two crucial concepts, Contributions, and Manifests, are introduced, operating like plugin systems known from other applications. These modules expose contribution points, which allow the introduction of extra functionalities in the module by delegating their execution to registered components. This delegation allows functionalities to be executed from implementations outside the module's primary responsibility.

In essence, Contributions establish well-defined interfaces for these delegated functionalities, which can be registered externally. On the other hand, Manifests serve as containers for dependency injection. They are responsible for registering the functionality requested from the Contributions and ensuring that the necessary functionalities are available and accessible when required. This way, the architecture maintains flexibility and modularity and eases the integration of new features while adhering to a clean separation of concerns.

B. *Multi Representation Support*

One of the main reasons for adhering to the reference architecture is to efficiently support various diagram representations, as UML encompasses numerous elements that can be visualized in different forms and shapes. Specifically, UML version 2 includes seven Structure Diagrams and seven Behavior Diagrams, which share some common primitive types. For instance, the Communication and Sequence diagrams largely share the same elements which are visually represented differently in both diagrams but saved similarly in the source models—because we are using the UML 2 Ecore resources.

Consequently, the system saves the active representation from the current source file to handle this diversity. This information also enables the system to load and utilize different features based on the specific diagram representation. Moreover, it also allows rendering elements depending on the representation while reusing the same components from the source models. This approach ensures greater flexibility and reusability within the system.

C. Deployment

BIGUML follows a client-server architecture, where distinct roles are assigned to each side. The server side implements language-specific functionalities, while the client solely focuses on presenting the information to users as already mentioned before. This approach is widely known as the language server protocol, which modern editors utilize. In the case of BIGUML, the server side is responsible for processing requests related to the UML editor by implementing language-specific functionality. On the other hand, the client side provides the user interface in VS Code and displays the UML diagrams and information to the users. The concrete components used on a running system are illustrated in Fig. 3 and defined in the following paragraphs.

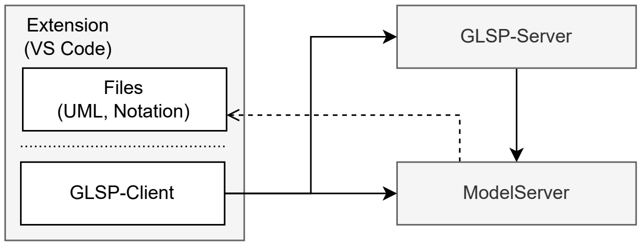

Fig. 3: BIGUML Deployment

a) Extension: BIGUML is distributed as an extension that users can easily install from the VS Code marketplace to enable a seamless UML modeling experience within VS Code. The extension's primary responsibility is handling the presentation of UML diagrams and enabling interaction with the diagram in the editor without getting involved in language-specific complexities. For diagram rendering and communication with the server, BIGUML relies on the GLSP-Client, which complements the GLSP-Server. Additional information will be discussed in more detail in Section III-B.

b) GLSP-Server: The GLSP-Server is a Java-based application utilizing the GLSP framework. It functions as the counterpart to the GLSP-Client and is responsible for tasks such as transforming the UML model into diagrams for rendering in the GLSP-Client and handling other triggered user actions in the editor. Notably, the GLSP-Server does not directly manage the source models; instead, it reads the models from the model server. Any modification requests concerning the source models are forwarded by the GLSP-Server to the model server. This separation of responsibilities ensures efficient and organized management of diagrams and source models.

c) Model server: The model server is responsible for managing all aspects of the source models and performing CRUD operations on them. It is the only part of the system capable of reading and understanding UML files stored on the disk and modifying them accordingly. For instance, when the GLSP-Client needs to render diagrams, it requests data from the GLSP-Server, which, in turn, retrieves the content from the model server. The GLSP-Server then returns the UML elements as a diagram to be rendered in the GLSP-Client. This division of responsibilities guarantees that the model server maintains complete control over the source models, consequently ensuring data integrity and proper management of UML files stored on the disk.

Apparent to the components visible, the BIGUML extension has a runtime requirement to function properly. Accordingly, after starting the extension in VS Code, it will automatically also start those servers in the background.

III. BIGUML IN USE

Next, we will present examples regarding BIGUML and explore functionalities that are currently available.

A. Example

To showcase the main features of the BIGUML tool, we show two examples, as seen in Fig. 4 and Fig. 5. The former is a simple class diagram for a lecture web service, and the latter is a use case diagram for an airport check-in. In detail, the BIGUML extension is structured into two parts. On the left side, we have a custom panel that integrates directly into VS Code and provides contextual functionalities based on the active diagram. This panel currently consists of two views: the *property view* (visible in Fig. 4) and the *outline view* (visible in Fig. 5). Accordingly, the property view displays properties and details of selected elements in the diagram, while the outline view provides an overview of the diagram in a tree structure. The rest of the editor is taken by the diagram, presented within a custom tab. In this tab, the GLSP-Client renders the diagram and offers additional custom functionalities to enhance the user experience for working with the diagram, such as the *tool palette* (visible on the right side of Figs 4 and 5).

B. BIGUML *Core Functionalities*

BIGUML utilizes the GLSP framework as the foundation. That means that the GLSP framework already provides the editor with a set of pre-built features. The GLSP framework provides all the fundamental means to interact with the diagram, including renaming of graphical elements via a double-click and deleting via shortcuts. Next, it exposes a tool palette which is a list of options to select the node or edge to create in the diagram. The GLSP framework also provides a command

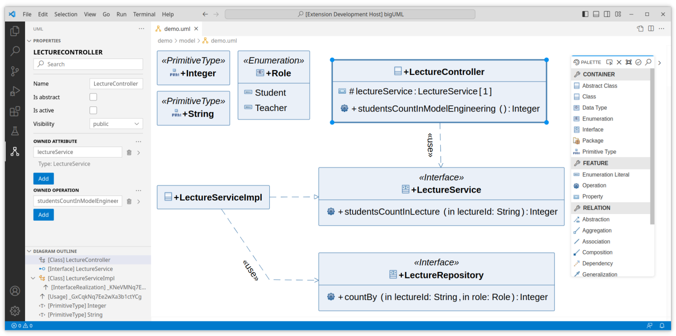

Fig. 4: BIGUML Class diagram

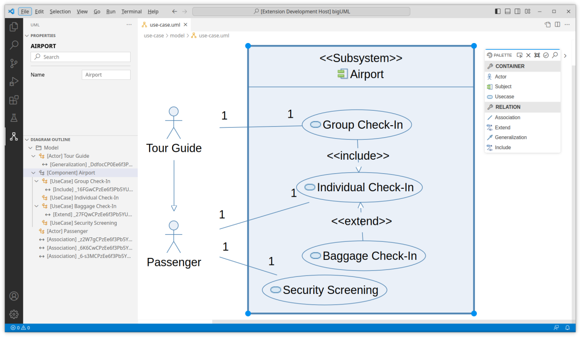

Fig. 5: BIGUML Use Case diagram

palette. The command palette is similar to other modern editors and allows the users to trigger specific commands. Similarly, it allows to execute commands with regard to the selected graphical element. More features exist, like diagram validation which shows markers on the graphical elements.

On top of those features, BIGUML introduces additional features that expand the capabilities beyond what the GLSP framework offers alone.

1) Wizard: The wizard in BIGUML serves as a useful way to create new diagrams and guides users through a step-by-step process. At present, users can specify the diagram's name and select the desired diagram type. Additionally, the wizard validates that the chosen path is valid to ensure an error-free diagram creation.

2) Property Palette: As previously mentioned, the property palette provides a means to modify the selected element outside the diagram. This feature proves valuable in cases where certain actions are not feasible through the graphical representation in the editor alone. For instance, many elements have multiple properties that users can customize. Displaying all of these properties directly on the diagram would make it cluttered and confusing. For this reason, the property palette offers a dedicated space where information that influences the source model can be accessed and edited without overloading the graphical editor.

Moreover, the property palette also serves as a convenient way to create new child properties, e.g., for enumerations and their enumeration literals. This fast and accessible method enables users to efficiently manage and customize properties without complicating the diagram's visual presentation.

3) Outline View: The outline view functions as an overview of all elements present in the diagram. It provides users with a quick way to navigate and select specific elements. Moreover, it is bi-directional, which means that the users can see the selected item in the diagram highlighted as an entry in the outline view as well. By offering this comprehensive view, users can easily keep track of the entire diagram's content, ensuring they do not lose sight of the overall structure. Overall, the outline view is valuable in clearly understanding the diagram's composition.

4) Contextual (Copy-) Paste: Another noteworthy feature is the server's ability to perform context-based copy-paste operations. Typically, when users select elements and perform a paste operation, all the selected entries are duplicated. However, in the case of graphical elements, it can happen that not all the necessary graphical elements have been selected by the user. In such instances, it is up to the server to determine how to handle the paste operation. This also means that the outcome can be influenced programmatically, in particular depending on the diagram representation (e.g., the active diagram type).

Consider the default implementation for duplicating an edge. If the source or target node is not selected, the server may choose to reuse the existing node the edge is connected to and duplicate only the rest. On the other hand, if all elements are selected, the server may duplicate all the interconnected elements together, creating new connections between them. This context-based approach ensures that the copy-paste operation adapts to the specific elements selected and maintains the semantic integrity of the diagram while providing a seamless user experience.

IV. Related Tools

A wide variety of UML modeling tools are available [2], and on the surface, they may seem similar, as shown in Table I. This section compares some of these UML tools to highlight their key differences because each of them allows users to create UML diagrams to some extent. A more comprehensive comparison to the related tools can be found in [2] and is out of scope of this tool paper.

Eclipse Papyrus[3] and StarUML[4] are both robust UML editors, each offering a native application for users. Papyrus is an industrial-grade open-source model-based engineering tool [10]. However, Papyrus can be complex, and its user interface may not be as intuitive compared to more modern tools. On the other hand, StarUML, which is also widely known, requires a fee for usage and is not open-source. Draw.io[5] is a general graphical creation tool that mainly focuses on the visual representation of diagrams and lacks emphasis on the underlying semantics as it offers basic support for UML. Meanwhile, PlantUML[6] is an advanced tool that enables diagram creation through a simple plain text language

[2]https://modeling-languages.com/uml-tools/
[3]https://eclipse.dev/papyrus/index.php
[4]https://staruml.io/
[5]https://www.drawio.com/
[6]https://plantuml.com/

TABLE I: Related Tools

	BIGUML	**Papyrus**	**PlantUML**	**StarUML**	**Draw.io**
Open Source	Yes	Yes	Yes	No (licensed)	Yes (with limitations)
Platform	VS Code	Native Application	Multi-Platform	Native Application	Web, VS Code
Input Type	Graphical	Graphical	Text	Graphical	Graphical
UML Focus	Yes	Yes	Yes	Yes	No
LSP	Yes	No	No	No	No

specification. Yet, it is not possible for users to freely position the graphical elements according to the users' preferences.

Although BIGUML also aims to create UML diagrams, it sets itself apart by fully utilizing the Language Server Protocol and the available UML semantics. This approach makes it the first modeling tool to leverage LSP for UML and being distributed through the VS Code ecosystem. Using LSP, users can easily reuse the capabilities of the BIGUML language server in other editors like Theia or other platforms. Additionally, developers can define their diagrams in the same environment where they write their code.

Note that BIGUML is still under active development and may not be as powerful as well-established tools like Papyrus or StarUML. However, new features and diagram types are continuously being introduced in iterative updates.

V. Future Work

BIGUML is currently in its early release stage and is actively being developed. As the foundational components of the architecture are being prepared, students from TU Wien are working on developing various UML diagram types. However, these diagrams are still under review and need to be integrated into the extension gradually before they become accessible to external users. Additionally, the visual representation of these different diagram types is still a work in progress.

In the future, BIGUML plans to introduce smart features that enhance the user experience. These features may include automating the creation of elements, intelligently connecting nodes based on the context, and allowing users to further customize the visual rendering of the diagrams [11]. These enhancements are aimed at providing users with a more efficient and personalized UML modeling experience. Moreover, it is planned to allow the users to customize how the graphical elements are rendered and to allow the users to import UML models from other tools like Papyrus.

VI. Conclusion

In this paper, we introduced the BIGUML modeling tool, a promising approach to UML modeling within the realm of available UML tools. While there is a wide variety of UML editors, each with its strengths and limitations, BIGUML stands out by being flexible to extend by other developers, fully harnessing the LSP capabilities, and seamlessly integrating into the VS Code ecosystem[7]. BIGUML aims to provide a modeling tool that teachers can recommend and use in modeling courses according to their preferences. Future versions will provide richer functionalities to make the modeling experience smoother and more appealing. We are also working on a collaboration feature.

Overall, BIGUML emphasizes combining user-friendliness with powerful UML modeling capabilities through the GLSP framework to become a valuable tool for UML enthusiasts and developers. As technology evolves and user feedback shapes development, BIGUML aims to contribute significantly to the UML modeling landscape. A video showcasing BIGUML in use can be found at: https://youtu.be/GR-hSB0ZOfE.

Acknowledgments

We want to thank EclipseSource Vienna for the close collaboration and all the students of TU Wien who contributed to the development and evaluation of BIGUML.

References

[1] H. Ossher, A. van der Hoek, M. D. Storey, J. Grundy, and R. K. E. Bellamy, "Flexible modeling tools (FlexiTools2010)," in *32nd ACM/IEEE Int. Conf. on Software Engineering - Volume 2*, 2010, pp. 441–442.
[2] M. Ozkaya, "Are the UML modelling tools powerful enough for practitioners? A literature review," *IET Softw.*, vol. 13, no. 5, pp. 338–354, 2019.
[3] P. Pourali and J. M. Atlee, "An empirical investigation to understand the difficulties and challenges of software modellers when using modelling tools," in *21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. ACM, 2018, pp. 224–234.
[4] G. D. Carlo, P. Langer, and D. Bork, "Advanced visualization and interaction in GLSP-based web modeling: realizing semantic zoom and off-screen elements," in *25th International Conference on Model Driven Engineering Languages and Systems*. ACM, 2022, pp. 221–231.
[5] H. Metin and D. Bork, "On developing and operating glsp-based web modeling tools: Lessons learned from bigUML," in *Proceedings of the 26th International Conference on Model Driven Engineering Languages and Systems, MODELS 2023*. IEEE, 2023.
[6] D. Bork, P. Langer, and T. Ortmayr, "A vision for flexibile glsp-based web modeling tools," *CoRR*, vol. abs/2307.01352, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2307.01352
[7] "Microsoft language server protocol specification," https://microsoft.github.io/language-server-protocol/specifications/specification-current/, accessed: 13.04.2023.
[8] "Microsoft language server protocol implementations," https://microsoft.github.io/language-server-protocol/implementors/servers/, accessed: 13.04.2023.
[9] R. Rodríguez-Echeverría, J. L. C. Izquierdo, M. Wimmer, and J. Cabot, "Towards a language server protocol infrastructure for graphical modeling," in *21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. ACM, 2018, pp. 370–380.
[10] A. Lanusse, Y. Tanguy, H. Espinoza, C. Mraidha, S. Gerard, P. Tessier, R. Schnekenburger, H. Dubois, and F. Terrier, "Papyrus UML: an open source toolset for MDA," in *5th European Conference on Model-Driven Architecture Foundations and Applications*, 2009, pp. 1–4.
[11] G. D. Carlo, P. Langer, and D. Bork, "Rethinking model representation - A taxonomy of advanced information visualization in conceptual modeling," in *41st International Conference on Conceptual Modeling*. Springer, 2022, pp. 35–51.

[7]https://tinyurl.com/biguml-vscode