bigArchiMate – A new Open and Extensible ArchiMate Modeling Tool

Phillip Winder[1], Martin Fleck[1] and Dominik Bork[1,*]

[1]TU Wien, Business Informatics Group, Favoritenstrasse 9-11, 1040 Vienna, Austria
[2]EclipseSource Services GmbH, Schwindgasse 20/2-3, 1040 Wien

Abstract

ArchiMate is a widely adopted framework that provides a standardized modeling language for representing the structure and behavior of enterprise architectures. However, traditional ArchiMate modeling tools are often proprietary, platform-dependent, and limited in extensibility. This paper introduces bigArchiMate , a flexible, open-source ArchiMate modeling tool built on Eclipse Theia, leveraging the Graphical Language Server Protocol (GLSP) and the Langium language engineering tool to provide an integrated modeling experience. bigArchiMate supports textual, graphical, and form-based modeling, ensuring synchronization across different perspectives through a shared semantic model. Key features include a package-based project system, a custom model service facade, and automated syncing mechanisms for multi-client environments. As the first open-source ArchiMate tool to incorporate GLSP, bigArchiMate offers an extensible and user-friendly alternative to traditional ArchiMate modeling solutions.

Keywords

ArchiMate, Enterprise Architecture, Blended Modeling, GLSP, Langium, DSL, Eclipse Theia, Web Modeling

1. Introduction

ArchiMate[1], maintained by The Open Group[2], is a standardized modeling language for enterprise architecture modeling, enabling the representation of business processes, applications, and technology infrastructure[1]. While widely used, traditional ArchiMate tools—often proprietary and desktop-based—suffer from limited extensibility, platform dependency, and a focus on singular graphical editing perspectives, neglecting textual or form-based approaches that could enhance usability across diverse workflows.

This paper presents bigArchiMate , an open-source tool designed to overcome these limitations by integrating textual, graphical, and form-based modeling within a web-based environment, built on Eclipse Theia[3]. Utilizing the Graphical Language Server Protocol (GLSP)[4] for graphical editing and the Langium[5] framework for textual modeling, bigArchiMate unifies these perspectives through a shared semantic model and introduces innovative features like a package-based project system and multi-client synchronization, positioning it as a forward-thinking solution for ArchiMate modeling.

The development of bigArchiMate follows recent research in the model engineering community investigating the creation and possibilities of tools enabling users to interact with a single model through different synchronized notations using current next generation frameworks and technologies [2, 3]. This type of blended modeling, as defined by [4], allows temporary inconsistencies, as changes in one view (e.g., deleting part of an ID in text) may momentarily break representations in another (e.g., the graphical view) and offers various benefits to domain experts using a tool implementing it.[5] In

*Corresponding author.
e1618953@student.tuwien.ac.at (P. Winder); mfleck@eclipsesource.com (M. Fleck); dominik.bork@tuwien.ac.at (D. Bork)
0000-0001-8259-2297 (D. Bork)

[1]https://pubs.opengroup.org/architecture/archimate3-doc/
[2]https://www.opengroup.org/archimate-forum/archimate-overview
[3]https://theia-ide.org
[4]https://eclipse.dev/glsp
[5]https://langium.org

addition to other recent research focusing on developing tools following the blended model approach [6, 7] bigArchiMate shall serve as yet another proof of concept as to how this method can potentially not only revolutionize the development of real world modeling tools, but also offer an improved user experience for domain experts using them. bigArchiMate is open source, its source code can be found on GitHub[6].

2. bigArchiMate Architecture

bigArchiMate is built using the Eclipse Theia Platform, an extensible cloud and desktop IDE framework for building modern integrated development environments that run in both desktop and cloud settings. Theia offers a flexible, modular architecture that allows developers to tailor and extend the IDE with custom features while providing core functionalities like a file explorer, text editor, integrated terminal, problems view and Visual Studio Code extension support. The general architecture of bigArchiMate consists of several parts that aim to share a common data model and support different editing capabilities on that model. To address the limitations of currently existing tools, bigArchiMate was developed with the following goals:

- Enable textual modeling via a domain-specific language (DSL) implemented in Langium.
- Support graphical editing through a GLSP-based diagram interface.
- Offer form-based editing via a custom UI and model server.
- Ensure live synchronization between perspectives using a centralized, versioned document store.

2.1. Textual Modeling with Langium

Textual modeling in bigArchiMate is enabled by the Langium framework which is deeply integrated with the Language Server Protocol (LSP), a standardized protocol designed to provide rich language features within development environments. LSP facilitates the separation of language-specific logic from the client user interface, enabling language tooling to be reused across editors.

Langium provides an LSP-compliant infrastructure that automatically generates a language server from a grammar specification. This design makes it easier to define domain-specific modeling languages while leveraging powerful IDE-like features such as code completion, validation, and real-time feedback. On application start, the bigArchiMate language extension initializes the Langium-based language server. The server scans the entire workspace, indexes documents, and constructs an abstract syntax tree (AST) for each valid ArchiMate file. The ASTs are persisted in an in-memory document store that becomes the authoritative source of truth for the rest of the system.

Furthermore bigArchiMate introduces a package manager mechanism that enables encapsulation, modular reuse, and versioning of ArchiMate models. Any directory with a package.json file is treated as a separate ArchiMate model. Dependencies between packages are resolved using npm-style semantics, allowing models to reference only those elements explicitly declared as dependencies. Moreover, using the familiar package.json format allows organizations to publish and share ArchiMate models over standard npm registries. The package manager parses and validates the package structure, ensures semantic isolation of components, and facilitates cross-package linking by creating a dependency graph.

2.2. Graphical Modeling with GLSP

bigArchiMate supports diagrammatic modeling using the Graphical Language Server Platform (GLSP), an open-source framework that extends the language server concept to graphical modeling, enabling a clean separation between the web-based graphical user interface and the underlying domain-specific modeling logic[8, 9]. By leveraging a JSON-RPC communication protocol similar to the Language Server Protocol, GLSP facilitates the development of modular, extensible, and synchronized diagram editors and modeling tools.[10]

[6]https://github.com/borkdominik/bigArchiMate

A distinctive architectural decision in bigArchiMate is to host the GLSP server within the same process as the Langium language server. This co-location strategy reduces communication overhead and allows shared access to services such as model indexing, validation, and change notifications. The GLSP client initiates the rendering process by requesting a diagram for a specific model file. The GLSP server, in turn, queries the Langium document store, converts the semantic model into a platform-independent view model representation, and sends it back to the client for rendering.

Graphical interactions, such as node creation, edge linking, or node deletion, are processed on the client and sent to the server as commands. The server interprets these commands and applies the changes to the in-memory AST, ensuring consistency with the textual representation. By separating semantic concerns (handled by Langium) from rendering logic (handled by GLSP), bigArchiMate adheres to the Separation of Concerns principle and ensures that each subsystem remains maintainable and independently extensible.

2.3. Form-Based Modeling

To further improve user experience, bigArchiMate introduces a form-based modeling interface. This interface is built on a custom RPC protocol layered over a dedicated Model Server, which exposes a high-level API for retrieving and updating model data. When a file is opened using the form-based editor, the client requests the semantic model from the model server, which serializes the AST into a form-optimized structure. The frontend then renders this structure into HTML forms, with each form section representing a specific element or property in the model. Changes made in the form (e.g., modifying a property value, adding child elements) are transmitted back to the model server. The server revalidates and updates the AST, preserving synchronization with the centralized document store. This interface is especially beneficial for users who do not wish to interact directly with text files.

2.4. Synchronization Across Perspectives

To synchronize the different modeling perspectives (textual, graphical, form-based) the central document store provided by Langium is leveraged as a shared global state accessible to all clients. The document lifecycle aligns closely with the expected behavior of the Language Server Protocol (LSP): an initial open call transfers the document content from the client to the server, establishing the server as the authoritative source of truth until the document is closed. Each client update modifies the server's in-memory representation and increments an internal version number.

In bigArchiMate this lifecycle is extended to support multi-client scenarios by requiring only the first open call to transfer document content—subsequent open calls are ignored, as the server already maintains the authoritative document state. Between open and close operations, clients can freely send updates. To enable live synchronization, every applied update is broadcast to all connected clients, along with metadata identifying the initiating client. This allows clients to selectively apply or discard updates based on their internal state, effectively preventing update cycles. A notable exception is the Monaco-based textual language client, which is updated directly by the server through the applyEdit request, as it cannot hook into the custom listener mechanism. Currently, the system supports full-model updates, where the complete model (or text) is transmitted and replaced in each client. Once the final client closes the document, the server clears the in-memory state and will only accept new updates after a subsequent open operation.

3. bigArchiMate tool demonstration

In the following section the terms model, concept, element, relationship and junction refer to their respective definitions in the ArchiMate Specification of The Open Group[7].

[7] https://pubs.opengroup.org/architecture/archimate3-doc/ch-Language-Structure.html

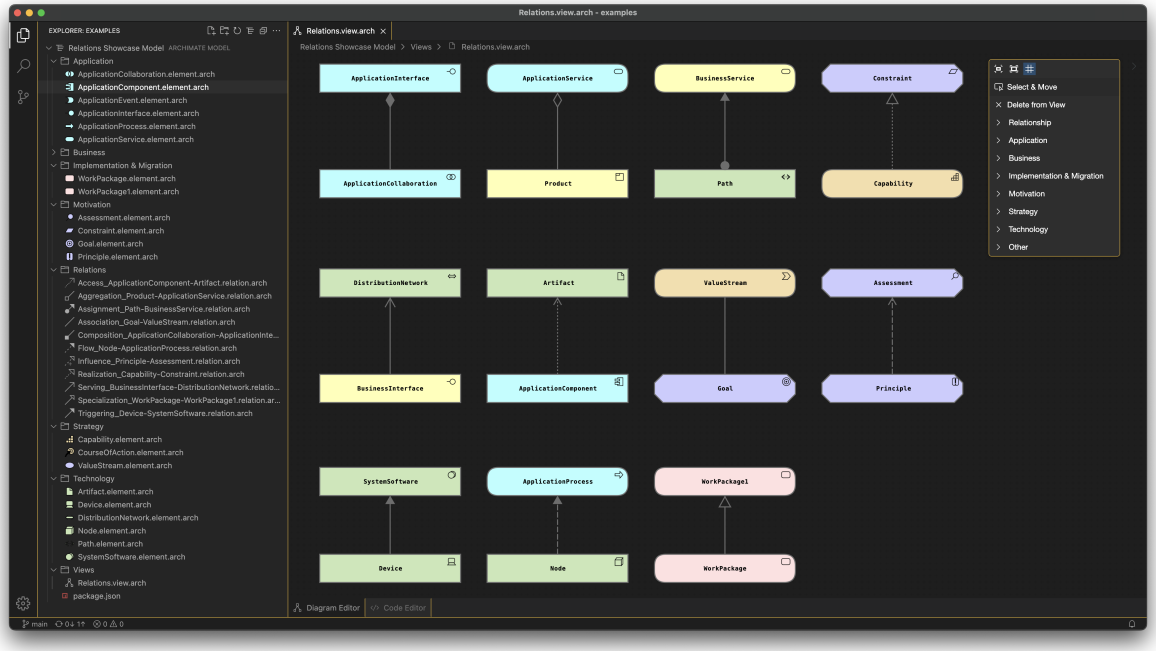

Figure 1: bigArchiMate user interface dark mode

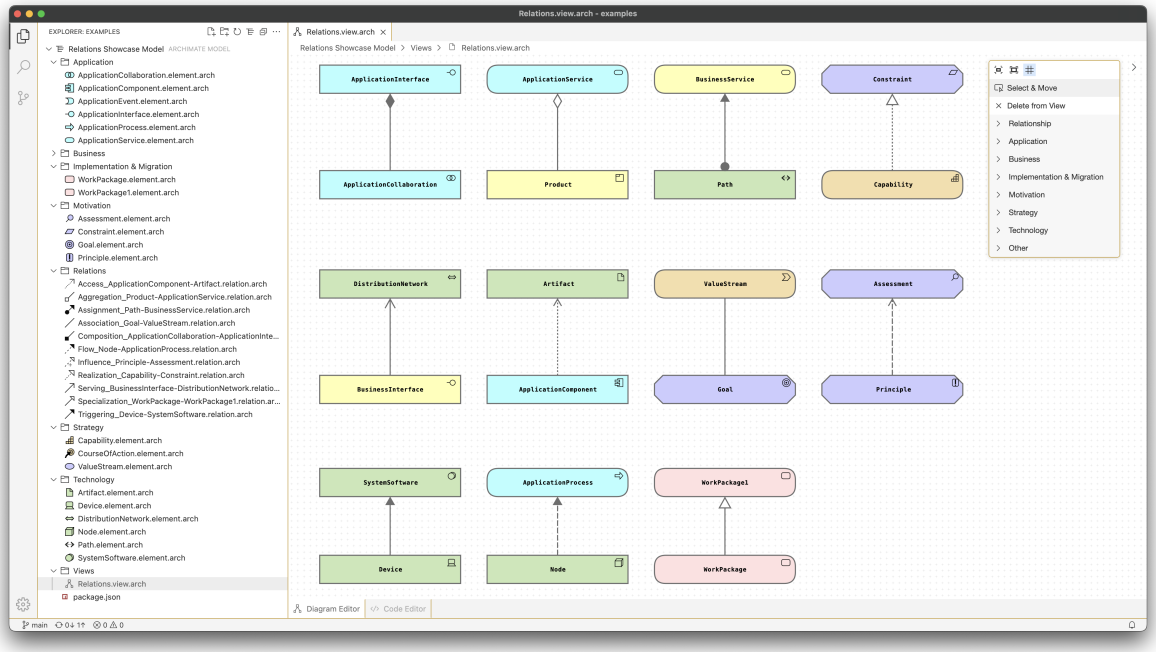

Figure 2: bigArchiMate user interface light mode

3.1. User Interface

bigArchiMate leverages several tools offered by the Eclipse Theia Platform providing a frontend that incorporates features like navigating and managing files in a file explorer, search and replace functionality, tracking file changes in a dedicated source control section and running shell commands directly from within bigArchiMate using an integrated terminal. Additionally, bigArchiMate of course extends the functionality to specific graphical, text, and form-based editors needed for ArchiMate

modeling. Different types of editors can be opened simultaneously for the same file(s). Depending on the content of the `.arch` file different types of editors are presented. For files that define concepts (i.e., they start with) *element*, *relation* or *junction* a form-based or text-based editor can be opened. For files defining views (i.e. they start with *view*), a graphical or text-based editor can be opened. A light and dark color scheme is offered.

3.2. Model structure

In bigArchiMate a model is represented as a folder containing a package.json file. The `name` and `version` defined in the `package.json` file of a model serve as a unique identifier for other models cross-referencing files from it by defining it as a dependency in their own `package.json` file. Each concept or view contained in this model is represented as a text file ending with the file extension `.arch`. Compound file extensions are used to determine the corresponding graphical, form-based or text-based editors that are presented to the user upon opening a concept or view (`.element.arch`, `.junction.arch`, `.relation.arch`, `.view.arch`). In the case of concepts the start of the filename by convention corresponds to its type and is used to determine the corresponding ArchiMate icon displayed in the file explorer. By default bigArchiMate automatically groups views and concepts of a model into certain pre-defined folders that relate to the layer or function of the respective concepts (`Application`, `Business`, `Implementation & Migration`, `Motivation`, `Other`, `Relations`, `Strategy`, `Technology`, `Views`) when they are added using the graphical editor of a view to provide a clear overview of the model. However, this structure is not necessary for a bigArchiMate model to work and could therefore be adapted as needed. A new model can be scaffolded using the *New Model...* command using the main menu bar, the command palette, the file explorer context menu or the file explorer toolbar. The file and text single source of truth nature of models in bigArchiMate makes it possible to use version control systems and manage them like other code repositories.

3.3. Model language

Concepts and views are defined in a YAML-like syntax that is processed by the Langium language server. The format was chosen because of its widespread popularity and adoption among the software engineering community. Additionally it provides high extensibility for additional language features in future versions of bigArchiMate . Syntax and cross-references are checked and verified. The editing experience is enhanced with features like syntax highlighting, code completion, refactoring, navigating to a symbol's definition, renaming and error and warning markers. The bigArchiMate language allows to define elements using the keyword *element*, relationships using *relation*, junctions using *junction*, and views using *diagram*.

All concepts and views must have an identifier (*id*) that is unique inside the model they are defined in. Additionally they can have an optional name, documentation and list of custom properties. A custom property must have an *id* and a *name* and can have a *value*. The scope of an identifier of a custom property is the file in which it is defined in. Views and relations use cross-references to other concepts. To cross-reference a concept from another model the referenced model has to be defined as a dependency in the current models package.json and its name prepended to the identifier of the concept separated by a dot. To neatly manage file names and identifiers bigArchiMate offers a naming convention mechanism. If the file name does not match the given id a warning is issued and the file can automatically be renamed using a context menu hint. If users want the corresponding ArchiMate concept icons to be shown as file icons the file name should start with the specific concept type. For files added using a diagram editor these conventions are automatically applied.

An **element** must have a *type* (which represents its element type according to the ArchiMate specification and is defined as an autosuggested enum in the bigArchiMate language) and a *name* (which is used as the label shown in the diagram editor of a view).

A **relationship** must have a *type* (which represents its relationship type according to the ArchiMate specification and is defined as an autosuggested enum in the bigArchiMate language). Additionally

properties *target* and *source* have to be cross-references to elements or junctions and are resolved by *id* as shown in Figure 3. The validity of relationships between concepts is checked and verified according to the relationship constraints described in the ArchiMate standard.[8]

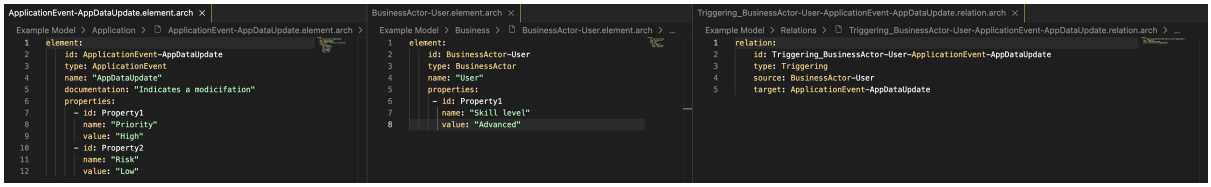

Figure 3: Example of two elements connected by a relationship using their respective *id*s to cross-reference them in the relationship definition

A **view** can have *nodes* and *edges*. Nodes and edges must have an *id*. A node must have a cross-reference to an *element* or a *junction* and has to define its coordinates (x and y) and its dimensions (*width* and *length*). An edge must have a cross-reference to the *relation* it represents and a source node (*sourceNode*) and target node (*targetNode*) accordingly. An edge can have additional routing points (*routingPoints*) which are represented as a list of coordinates (x and y) to adjust its appearance.

3.3.1. Editing a view

Views can be edited in a graphical- or text-based manner (Figure 4, Figure 5). The **Diagram Editor** of a view is the central tool for editing a model. It provides a tool palette that includes functionality for selecting, moving, creating and removing concepts. Additionally, concepts can be created using the context menu of the view. Only relationships that adhere to the constraints described in the ArchiMate standard can be created. Upon creating a new concept to the view a new file is added to the model and pre-filled with all mandatory properties. Additionally, the header of the tool palette provides buttons for resetting the viewport, fitting the diagram or a selected element to the size of its current bounds, and toggling a visual grid in the background of the diagram. Concepts are divided into togglable subsections. To improve user experience selecting a concept in a view opens its form-based editor in the Theia properties view (Figure 6). Existing concepts can be added to a view by dragging and dropping their respective file from the file explorer into the view. The **Code Editor** is the textual representation of the view and can be edited using the BIGARCHIMATE diagram language.

3.3.2. Editing a concept

Concepts can be edited in a form or text based manner (Figure 7, Figure 6). The **Form Editor** provides a form-based interface to edit all properties and custom properties of the concept. The **Code Editor** is the textual representation of the concept and can be edited using the BIGARCHIMATE language.

4. Related Tools

A wide variety of ArchiMate modeling tools are available, most of which are proprietary, closed source and paid. Currently there are 12 paid ArchiMate tools that are certified by The Open Group[9] and support different versions of the full ArchiMate specification. Some of these tools come with a stripped down free version and most of them are integrated into a full featured enterprise architecture management suite targeted at mature organizations with big teams leveraging the full ArchiMate specification and other enterprise architecture functionality. Archi[10] however is a popular free and open-source alternative distributed as a desktop application that is targeted toward all levels of Enterprise Architects and Modellers. It provides a low cost to entry solution to users who may be making their first steps in the

[8]https://pubs.opengroup.org/architecture/archimate3-doc/ch-relationships-Normative.html
[9]https://training.opengroup.org/tool-register/archimate
[10]https://www.archimatetool.com/

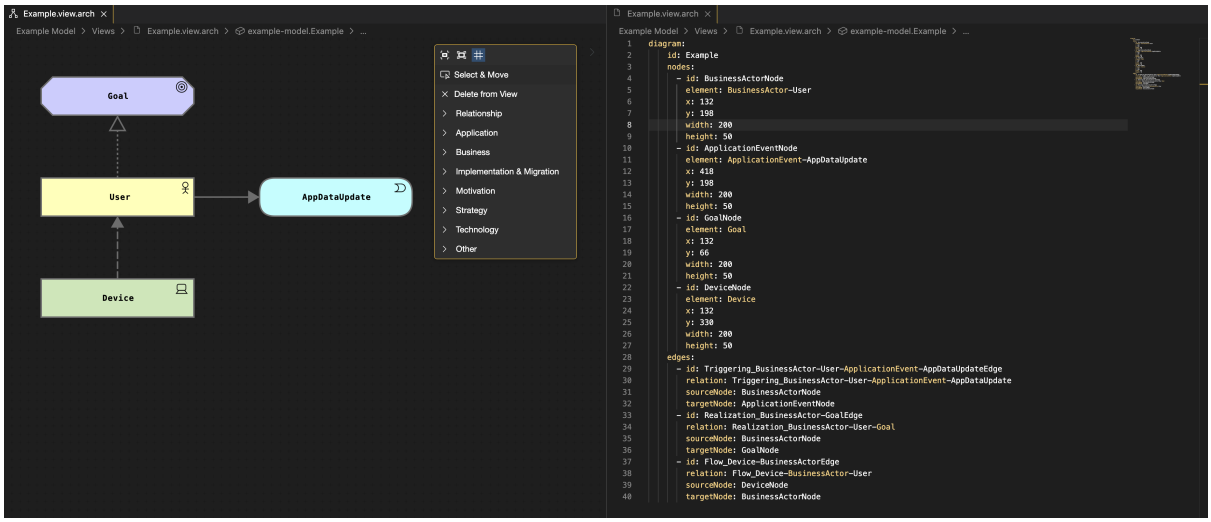

Figure 4: BIGARCHIMATE *Diagram Editor* and *Code Editor* open for a file *Example.view.arch*

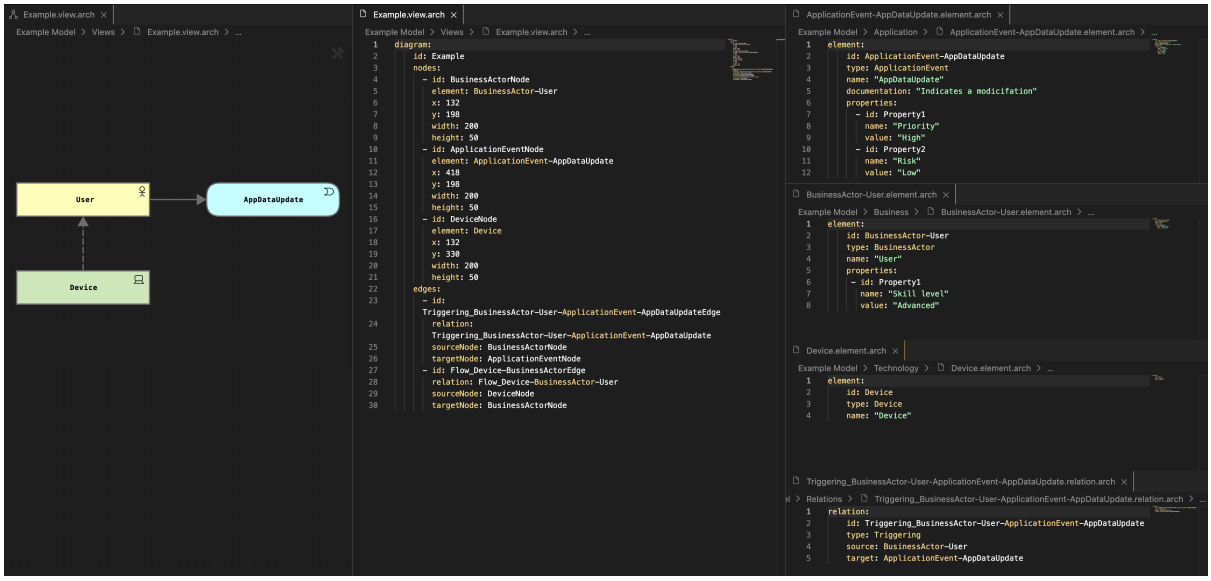

Figure 5: BIGARCHIMATE *Diagram Editor* and *Code Editor* open for a file *Example.view.arch*

and all *Code Editors* open for all concept files that are referenced in the view.

ArchiMate modelling language, or who are looking for a free, cross-platform ArchiMate modelling tool for their company or institution and wish to engage with the language within an Enterprise Architecture framework.[11]

None of the above mentioned tools, whether paid or free, closed or open source, support an integrated text editor in addition to a diagram editor let alone live synchronization between different types of editors. Some of the tools have a custom versioning concept but none of them come with built-in support for a version control system like git. Furthermore the distribution, extension and incorporation of (external) ArchiMate models/diagrams is mostly cumbersome and only works using imports/exports of ArchiMate models/diagrams, if at all. In contrast BIGARCHIMATE unifies textual DSL editing, GLSP-driven diagrams, and dynamic forms through a shared Langium document store. This document-based approach makes versioning using a version control system work out-of-the-box. Furthermore, BIGARCHIMATE is built as a modern web application leveraging state-of-the-art technologies enabling a modern User Interface and multiple themes.

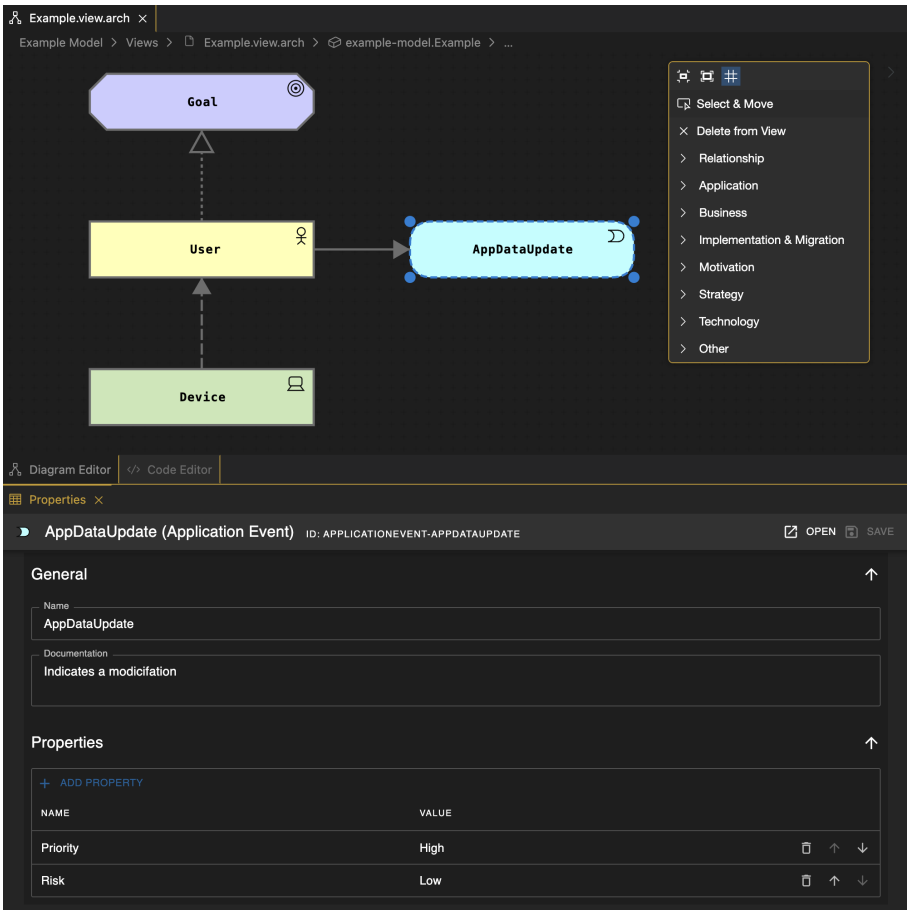

Figure 6: bigArchiMate *Diagram Editor* open for a file *Example.view.arch* and selected element *AppDataUpdate*

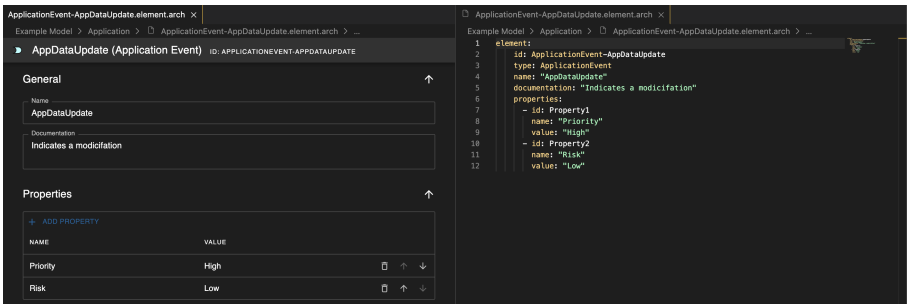

Figure 7: bigArchiMate *Form Editor* and Code Editor open for a file *ApplicationEvent-AppDataUpdate.element.arch*

5. Future Work

bigArchiMate is currently in its early release stage and is actively being developed and still missing some features defined in the ArchiMate standard. In its current version, bigArchiMate can be deployed as a browser application or as a standalone Electron application. However the language server powering the bigArchiMate language can already be deployed as a Visual Studio Code extension. Enhancing the extension with all other graphical and form-based features currently implemented using Theia could be of interest to further research. Furthermore bigArchiMate can be used as a reference for future tools implementing a similar blended modeling approach and serve as a starting point for many more domain specific modeling tools to come.

6. Conclusion

bigArchiMate represents a novel approach to not only ArchiMate enterprise modeling, but also modeling in general by combining textual, graphical, and form-based editing in a single toolchain. Its unified architecture powered by Langium, GLSP, and a custom model server enables robust synchronization and extensibility. Due to its open source nature, bigArchiMate is freely available and modifiable. Its development is also meant to be a proof of concept to inspire other tools combining graphical, textual and other forms of editing to use a similar approach.

7. Acknowledgments

We want to thank EclipseSource Vienna[11] for the close collaboration and all the students of TU Wien who contributed to the development and evaluation of bigArchiMate . We also gratefully acknowledge CrossBreeze[12] for inspiring the architecture and combination approach of bigArchiMate through their CrossModel[13] project.

References

[1] M. M. Lankhorst, H. A. Proper, H. Jonkers, The architecture of the archimate language, in: T. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, P. Soffer, R. Ukor (Eds.), Enterprise, Business-Process and Information Systems Modeling, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 367–380.

[2] I. David, M. Latifaj, J. Pietron, W. Zhang, F. Ciccozzi, I. Malavolta, A. Raschke, J.-P. Steghöfer, R. Hebig, Blended modeling in commercial and open-source model-driven software engineering tools: A systematic study, Software and Systems Modeling 22 (2022) 415–447.

[3] A. Lencses, Combining textual and graphical modeling with next generation frameworks, 2024.

[4] F. Ciccozzi, M. Tichy, H. Vangheluwe, D. Weyns, Blended modelling - what, why and how, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2019, pp. 425–430.

[5] L. Addazi, F. Ciccozzi, P. Langer, E. Posse, Towards seamless hybrid graphical–textual modelling for uml and profiles, in: A. Anjorin, H. Espinoza (Eds.), Modelling Foundations and Applications, Springer International Publishing, Cham, 2017, pp. 20–33.

[6] P.-L. Glaser, G. Hammerschmied, V. Hnatiuk, D. Bork, The bigER modeling tool, 2022.

[7] P.-L. Glaser, D. Bork, The biger tool - hybrid textual and graphical modeling of entity relationships in vs code, in: 2021 IEEE 25th International Enterprise Distributed Object Computing Workshop (EDOCW), 2021, pp. 337–340.

[8] D. Bork, P. Langer, T. Ortmayr, A vision for flexibile GLSP-based web modeling tools (2023).

[9] D. Bork, P. Langer, Language server protocol: An introduction to the protocol, its use, and adoption for web modeling tools, Enterprise Modelling and Information Systems Architectures (EMISAJ) (2023) Vol. 18 (2023).

[10] H. Metin, 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, pp. 129–139.

[11] Archi: Archimate modelling tool, 2025. URL: https://github.com/archimatetool/archi, accessed: 2025-04-10.

[11] https://eclipsesource.com

[12] https://crossbreeze.nl

[13] https://github.com/CrossBreezeNL/crossmodel