

The BIGER Modeling Tool

Philipp-Lorenz Glaser¹, Georg Hammerschmied¹, Vladyslav Hnatiuk¹ and Dominik Bork^{1,*}

¹TU Wien, Business Informatics Group, Favoritenstrasse 9-11, 1040 Vienna, Austria

Abstract

This paper introduces the first major release of the BIGER modeling tool. BIGER offers various features for flexibly specifying and visualizing Entity Relationship (ER) data models. Within the Visual Studio (VS) Code IDE, the tool enables hybrid modeling through a textual editor and a graphical editor to display and modify the textual and graphical ER model, respectively. Both editors are realized with a custom language to specify ER elements and allow multi-notation support (currently Bachman, Chen, Crow's Foot, Min-Max, and UML). The BIGER modeling tool incorporates the Language Server Protocol and is based on web technologies, which makes the tool platform-independent and easily extensible. We present the newest extensions of BIGER, i.e., multi-notation support and improved edge routing.

Keywords

Entity Relationship, Modeling, Tool, Language Server Protocol, Sprotty, Code generation

1. The BIGER Modeling Tool – A VS Code Extension

ER modeling [1] is a common practice to conceptualize relational data, generally accomplished through the use of modeling tools. However, many of today's modeling tools do not properly capture the underlying semantics of ER models, resulting in erroneous modeling, due to lack of validation. Furthermore, there are various limitations in current modeling tools, such as, e.g., support only for graphical modeling, leading to decreased efficiency when creating ER models.

In this work, we present the BIGER modeling tool, deployed as a free extension to the Visual Studio (VS) Code IDE – at the time of writing with more than 800 downloads/users. BIGER allows hybrid ER modeling with a textual and graphical editor. This combined approach enables to take advantage of both, stability and efficiency when using the textual representations, and improved comprehensibility and cross-stakeholder communication when creating models graphically [2]. Furthermore, the tool allows generating SQL table schemas from the created models. First conceptual ideas and prototypical developments regarding BIGER have been reported in [3]. In the work at hand, we report on our ongoing efforts in extending the functionality provided by BIGER and increasing its usability. Concretely, we report on the following core extensions: *i) Multi-Notation Support* for creating models in a variety of the most prevalent ER notations, and *ii) a new Edge Router* for an improved graphical representation.

ER'2022 Forum and PhD Symposium, October 17-20, 2022, Online


*Corresponding author.

✉ philipp-lorenz.glaser@tuwien.ac.at (P. Glaser); georg.hammerschmied@hotmail.de (G. Hammerschmied); e01613669@student.tuwien.ac.at (V. Hnatiuk); dominik.bork@tuwien.ac.at (D. Bork)

🆔 0000-0002-0710-8052 (P. Glaser); 0000-0001-8259-2297 (D. Bork)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

In the following, we first describe the architecture of the tool and provide a demonstration of its main features. We then elaborate the above-mentioned extensions in more detail before we conclude this paper.

Tool Architecture BIGER is based on the Language Server Protocol (LSP) and the diagramming framework Sprouty¹ with a client and a server component. The server is implemented as a language server with Xtext² that communicates through the LSP to the extension (client) to enable rich-text editing support. The server is further enhanced with Sprouty to support graphical language features, e.g., with a diagram generator for the graphical model. The client-side contains a language client that communicates with the server and integrates Sprouty with VS Code to render the diagram in a web view. A web view manager is responsible for managing the web view where Sprouty-specific components are located, such as, e.g., SVG views or CSS styling. Fig. 1 shows a high-level view of the BIGER architecture – for a detailed view, see [4].

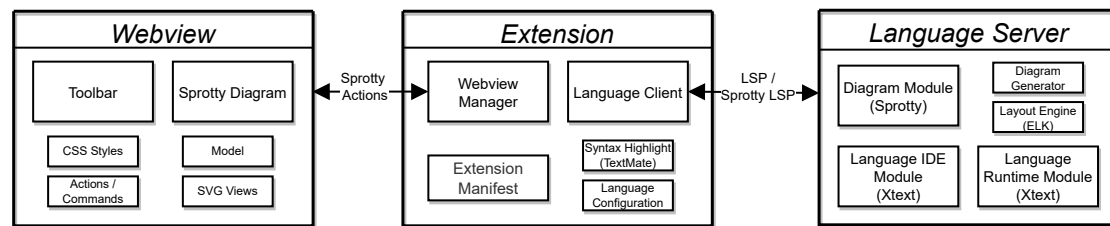


Figure 1: Architecture of the BIGER modeling tool

Tool Demonstration ER models are created in text files ending with `.erd` which, when opened in VS Code, automatically activate the extension. The tool mainly consists of three components for modeling: *i*) the *textual editor* (Fig. 2 left) to specify the ER models through a textual Domain-specific Language including rich-text editing support (e.g., syntax highlighting, code completion, validation), *ii*) the *diagram view* (Fig. 2 center) that renders an ER diagram corresponding to the specification in the textual file that is also synchronized on text changes and can be partly used as a graphical editor, and *iii*) the *code generator* (Fig. 2 right) for transforming the conceptual ER models into a logical schema for a relation database by generating SQL tables. The ER model in the figure features a basic example of a university database, where students take exams of courses that include multiple lectures and are graded by instructors. As can be seen in the textual editor, various keywords are used to classify model elements, a complete specification of the language can be seen in the language guide on GitHub³.

Multi-Notation Support The first major extension to the tool is multi-notation support to enable modeling in different ER variations. The notation can be changed textually or graphically and if not specified, a default notation is used (see Figure 2). Besides the default notation, BIGER supports many popular ER notations including *Bachman*, *Chen*, *Crow's Foot*, *Min-Max*, and *UML*. To only allow the characteristics of one notation at a time, the language server validates the textual model and informs the user whether the model contains an error or the specified cardinality does not match the notation. If the model is valid, a new diagram representation that

¹Sprouty: <https://github.com/eclipse/sprouty>

²Xtext: <https://www.eclipse.org/Xtext/>

³BIGER language guide: <https://github.com/borkdominik/bigER/wiki/Language>

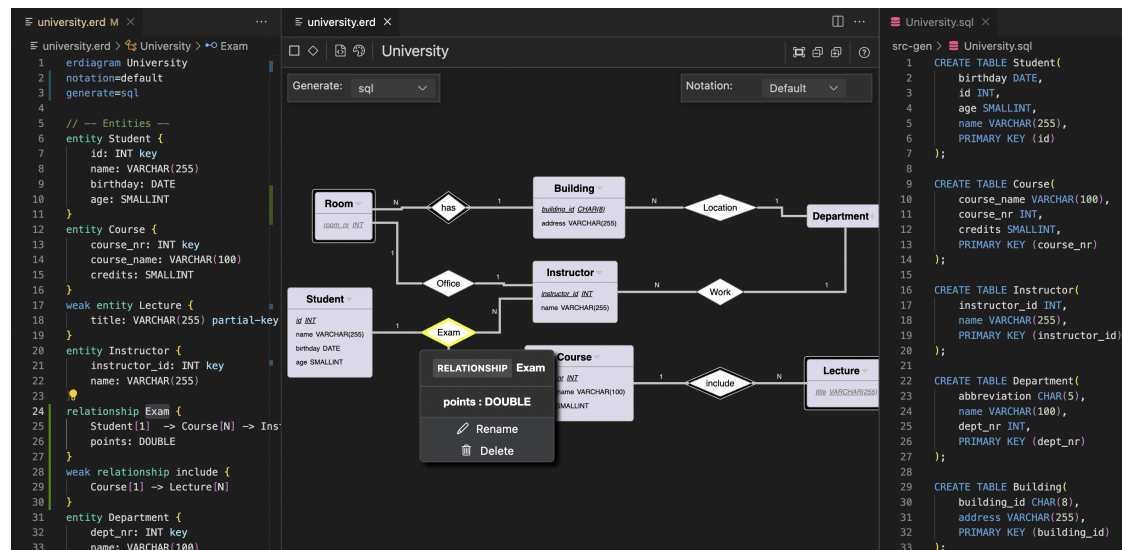


Figure 2: BIGER tool in VS Code with the main views open

entails notation-specific data is generated and sent to the client for rendering. In the web view on the client, the data is extracted, the cardinalities interpreted, and additional SVG elements are rendered on the edge if required by the specified notation and cardinality. An overview of the different notations and how they are rendered in BIGER can be seen in Table 1 including the textual syntax that is used to specify the cardinalities of relationships.

Notation	Textual Syntax	Diagram Representation
Bachman	$A[0], A[0+], A[1], A[1+]$	
Chen	$A[1], A[N], A[M]$	
Crow's Foot	$A[0+], A[1], A[1+], A[?]$	
Min-Max	$A[\min, \max]$ or $A[\min, *]$	
UML	$A[\text{num}], A[\min..\max]$ or $A[\min..*]$	

Table 1

Overview of the multiple notations currently supported by BIGER

Improved Edge Router An additional extension to the tool is the improved routing of edges in the diagram. Previously, the default router provided by Sprouty was used, which worked well in most cases, but had the disadvantage of routing each edge independent of other model elements, resulting in intersections between nodes and other edges (see *University* entity in Fig. 3 left). This problem was solved by replacing the router with a Libavoid router from the

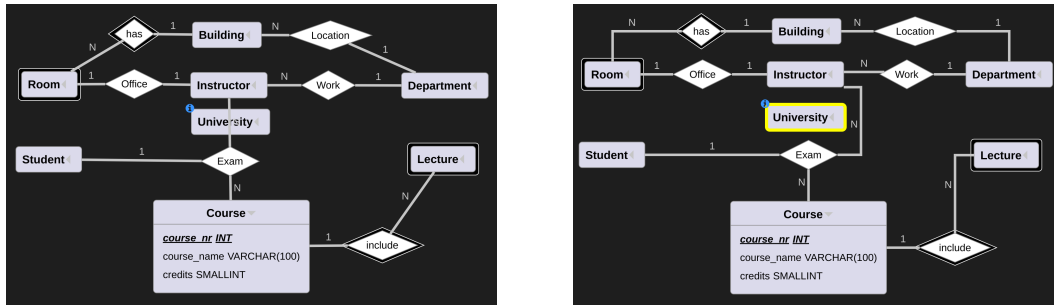


Figure 3: Old (left) versus improved (right) layout of the diagram model in BIGER

sprotty-libavoid-routing package⁴. The libavoid library is a well-known open-source solution for diagram routing including a feature for avoiding obstacles (i.e., other model elements). With the new router, manual changes of edges are not needed anymore and the routing of edges is completely automated. Two further benefits of the replacement include the use of orthogonal edges as opposed to more inconsistent polyline edges, and an overall improved behavior when nodes are being manually moved in the diagram (see Fig. 3 right).

2. Conclusion

We have presented the centerpieces of the BIGER modeling tool, proceeding our work in [3]. BIGER is the first ER modeling tool that is freely available as an extension to the widely used VS Code IDE, supporting hybrid ER modeling using a textual and a graphical editor that are always synchronized. The tool moreover supports five widely used ER notations and a code generator that transforms the ER models into SQL code. More details on the process of developing Sprotty-based modeling tools in general and the BIGER tool in particular are provided in [4]. We believe BIGER can be very valuable for the ER community and educators in database design courses. BIGER can be downloaded via the VS Code Marketplace⁵, the source code is available here⁶.

References

- [1] P. P.-S. Chen, The entity-relationship model—toward a unified view of data, *ACM Trans. Database Syst.* 1 (1976) 9–36.
- [2] J. Cooper, D. Kolovos, Engineering hybrid graphical-textual languages with sirius and xtext: Requirements and challenges, in: *22nd International Conference on Model Driven Engineering Languages and Systems Companion*, 2019, pp. 322–325.
- [3] P.-L. Glaser, D. Bork, The bigER Tool - Hybrid Textual and Graphical Modeling of Entity Relationships in VS Code, in: *25th International Enterprise Distributed Object Computing Workshop, EDOC Workshop 2021, IEEE, 2021*, pp. 337–340.
- [4] P.-L. Glaser, Developing sprotty-based modeling tools for vs code, 2022. URL: <https://model-engineering.info/publications/theses/thesis-glaser.pdf>, bachelor thesis at TU Wien.

⁴sprotty-libavoid-routing on GitHub: <https://github.com/Aksem/sprotty-routing-libavoid>

⁵<https://marketplace.visualstudio.com/items?itemName=BIGModelingTools.erdigram>

⁶<https://github.com/borkdominik/bigER>