The BIGER Tool - Hybrid Textual and Graphical Modeling of Entity Relationships in VS Code

Philipp-Lorenz Glaser and Dominik Bork

In:

2021 IEEE 25th International Enterprise Distributed Object Computing Workshop (EDOCW), pp. 337 – 340.

Final version available via DOI:
https://doi.org/10.1109/EDOCW52865.2021.00066

The BIGER Tool – Hybrid Textual and Graphical Modeling of Entity Relationships in VS Code

Philipp-Lorenz Glaser
TU Wien, Business Informatics Group
Favoritenstrasse 11, 1040 Vienna, Austria
Email: philipp-lorenz.glaser@tuwien.ac.at

Dominik Bork
TU Wien, Business Informatics Group
Favoritenstrasse 11, 1040 Vienna, Austria
Email: dominik.bork@tuwien.ac.at

Abstract—**The Entity Relationship model is the de-facto standard for data modeling and has been in use for a long time already. This popularity also led to the development of various tools that support ER modeling. However, these tools are often inflexible, proprietary, constrained to specific platforms, and lack advanced features like (SQL) code generation. This paper introduces the BIGER modeling tool. BIGER offers various features for flexibly specifying and visualizing conceptual ER data models. Within the VS Code IDE, the tool enables hybrid and interactive modeling through a textual editor with a custom language to specify ER elements and an accompanying view to display and modify the graphical ER model. The BIGER modeling tool is one of the first tools to incorporate the Language Server Protocol and to be distributed through the VS Code ecosystem. Due to its web technology-based architecture, it is platform-independent and easily extensible.**

Index Terms—**Data modeling, Modeling tool, Code generation, Hybrid modeling, VS Code, Language Server Protocol**

I. Introduction

The Entity-Relationship (ER) Model, first introduced by Chen in 1976, is a simple, yet very powerful language to conceptualize data models [1]. By using the core concepts of *Entities*, *Relationships*, and *Attributes*, objects and their associated properties can be put into relation. These objects, properties, or relations are part of a specific knowledge domain and can be specified to fulfill certain types of constraints through additional classifications. ER Model elements are represented visually in an *ER Diagram*, which has been adopted to different notations, depending on the respective domain. There have been various extensions to the classical model as well, such as the Extended Entity-Relationship (EER) Model, to include concepts such as generalization or specialization [2]. Albeit its long history, ER modeling is still a matter of ongoing research, see e.g. [3]

There are numerous graphical modeling tools for ER Diagrams, however, a combination with a textual modeling language might yield more efficiency in creating the models and might attract a wider user base. Textual modeling takes advantage of the various assisting features modern code editors can provide, e.g., finding references or auto-complete. Traditionally, these features had to be implemented separately with specific implementations for each individual editor. By introducing the Language Server Protocol (LSP)[1] these features become standardized and allow the reuse of a single language server for different clients (i.e., editors). Numerous popular IDEs like Eclipse, Sublime Text, or VS Code already support the protocol. The LSP, however, only addresses textual languages, proper support for graphical language features still has to be established, for first ideas, see e.g., [4].

[1]https://microsoft.github.io/language-server-protocol/

With the paper at hand, we show, to the best of our knowledge, the first realized blended modeling tool that realizes a language server and that is being deployed as an extension through the VS Code ecosystem. Thereby, we contribute first insights into the research directions of *Diagrammatic parsing* and *Architectures for blended modeling user interfaces* [5, p. 430], through the implementation of the resulting BIGER tool. This research also fits nicely into the current research stream of hybrid (or blended) modeling [5], [6].

II. The BIGER Tool

The aim and scope of this work is to enable hybrid textual and graphical modeling for Entity Relationship diagrams. Hybrid approaches to modeling are not new in itself, cf. e.g., [7] for UML and [8] for process modeling. They aim to be inclusive for different preferences and stakeholders involved modeling projects. Moreover, different purposes of modeling might be supported more efficiently by a graphical or a textual view on the model.

Considering that ER models are primarily in use by people who are already familiar with textual programming languages and given the benefits of efficiency and stability, the BIGER tool uses the textual representation as its leading model. Consequently, the graphical representation always follows the elements specified textually, and all sorts of editing actions modify the text first before changing the graphical model. This choice also enables the use of the LSP, even if an action is initiated within the graphical view, allowing for improved portability of the tool and standardized communication between the components (see Fig. 1).

A. Features

The BIGER modeling tool consists of three core components, a **textual editor** for the specification of ER models with a textual language, a corresponding **Diagram View** to display and interact with the model graphically, and a **Code Generator** to generate SQL statements of a database schema.

Synchronization between all the components is performed automatically, however, editing the generated SQL code does not change the underlying model. The rest of the section provides a brief overview of available features of BIGER, for a more complete list refer to the GitHub repository[2] or the VS Code Marketplace page[3] of the extension.

The textual editor with its language gets activated once a `.erd` file is opened. Within such an opened file, entities and relationships can be freely specified with attributes included within curly brackets. The textual language is designed to cover the classical concepts of ER models. Table I shows the mapping of the ER concepts to the corresponding textual concrete syntax. On the left side of Fig. 2, the use of the textual concrete syntax in our running example is shown. The editor of course also uses syntax highlighting to easily differentiate the ER language concepts from the domain-specific terms of the current model and includes numerous additional rich text editing features, such as *Validation*, and *Hyperlinking*.

Once the diagram view is opened, the model elements are transformed and a graphical ER Diagram is rendered. The diagram view also offers additional features to customize the view or modify the underlying model through the toolbar or by interacting with elements within the diagram directly, e.g., to create new entities and relationships or to edit attribute values.

B. Architecture

Fig. 1 shows the architecture of the BIGER tool which conforms to the requirements of the underlying technologies used for the implementation. This section explains the general responsibilities of the components and the reason behind design choices on a more abstract level, while the next subsection provides more insights into concrete technologies.

BIGER is realized in a client-server architecture. The server-side, implemented as a language server, provides language-specific functionality for both the textual as well as the graphical model. The syntax and semantics of the textual language are realized by specifying a grammar and implementing additional *Validation* and *Scoping*. The Code Generator is also included on the server side. For the graphical model an additional Diagram Server is added, which generates and synchronizes the diagram elements with the current textual model. The Layout Engine adds auto-layout to the diagram, together with additional implementations for graphical features.

Activation events in the extension define when the language client is activated, establishing the communication between client and server. The extension also contains configuration files for syntax highlighting, language configurations (e.g. bracket matching) and the extension manifest containing meta-data, dependencies, and VS Code specific contribution points.

In addition to the extension, which is responsible for the textual editor, the webview is also part of the client side and renders the generated diagram elements in a custom view next to the textual editor. Actions on the diagram are first passed

[2]https://github.com/borkdominik/bigER

[3]https://marketplace.visualstudio.com/items?itemName=BIGModelingTools.erdiagram

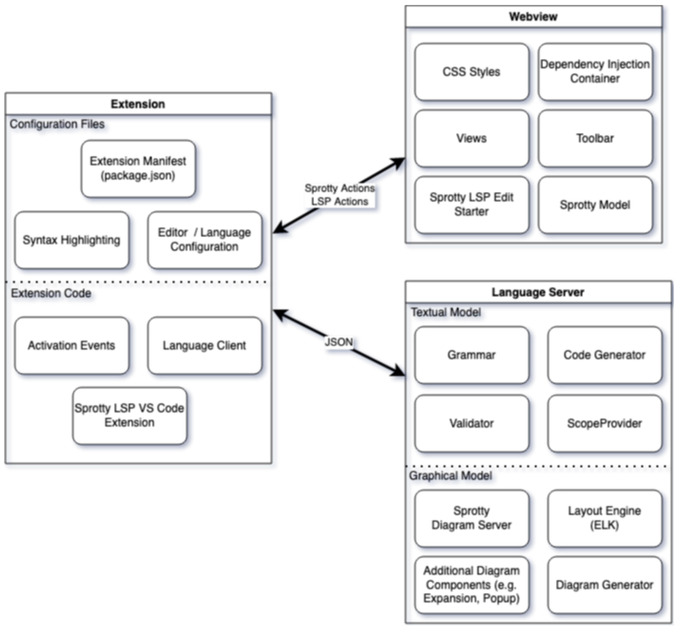


Fig. 1. Architecture of the BIGER Tool

TABLE I
TEXTUAL CONCRETE SYNTAX OF BIGER

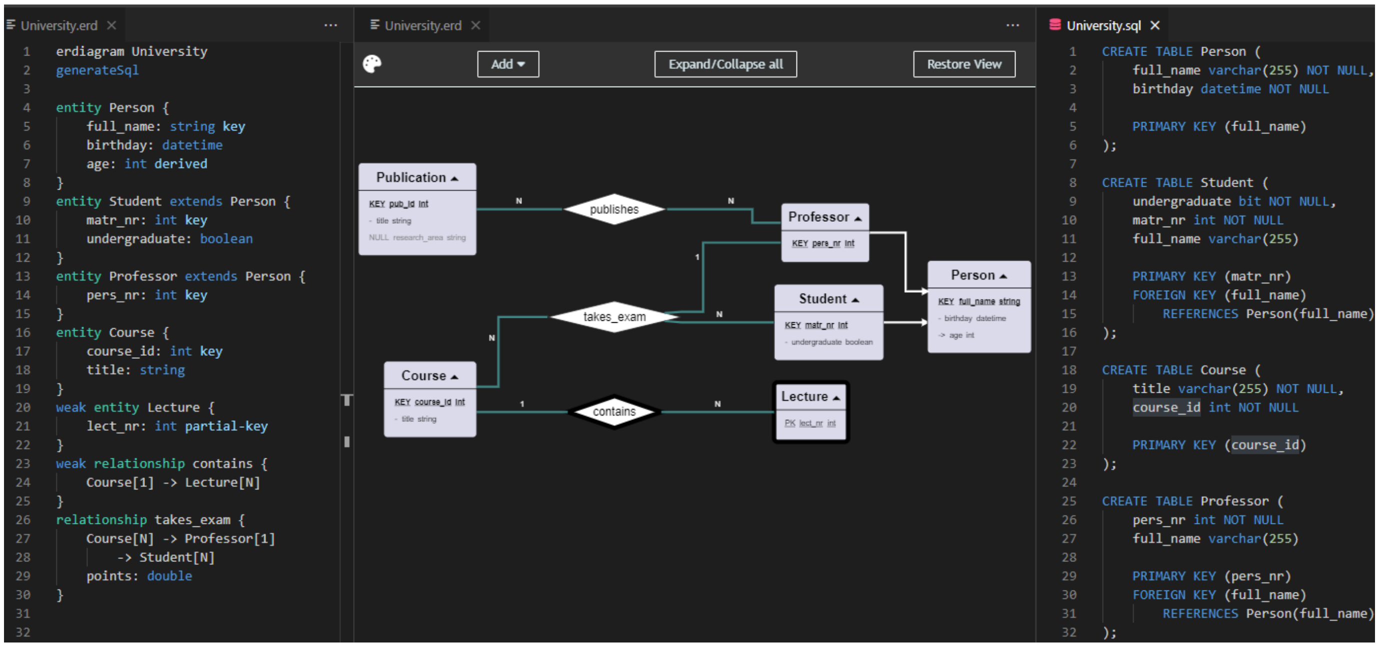
ER Concept	Textual Concrete Syntax
Strong Entity	`entity`
Weak Entity	`weak entity`
Inheritance	`A extends B`
Binary Relationship	`A -> B`
Recursive Relationship	`A -> A`
Ternary Relationship	`A -> B -> C`
Basic Cardinality	`A[1] -> A[N]`
Custom Cardinality Notation	`A["0..1"] -> A["1..N"]`
Simple Attribute with data type	`attribute: datatype`
Optional / Nullable	`optional`
Derived	`derived`
Multi valued	`multi-valued`
Primary key	`key`
Partial key	`partial-key`

to the extension, which either handles the actions on the client side or propagates them to the server.

C. Technical Realization

The language together with its textual editor features is implemented with the Xtext[4] language workbench. Xtext only requires a grammar file to specify the syntax of the ER language and how it is mapped to a semantic model. This also generates a language server with default implementations for various language features, which can be communicated to in form of LSP messages. The grammar for the BIGER language cannot be shown here given the limited space, but is is visible from the open Github repository[2]. To add further customization's to the tool, the default Validator and

[4]https://www.eclipse.org/Xtext/



```
1   erdiagram University
2   generateSql
3
4   entity Person {
5       full_name: string key
6       birthday: datetime
7       age: int derived
8   }
9   entity Student extends Person {
10      matr_nr: int key
11      undergraduate: boolean
12  }
13  entity Professor extends Person {
14      pers_nr: int key
15  }
16  entity Course {
17      course_id: int key
18      title: string
19  }
20  weak entity Lecture {
21      lect_nr: int partial-key
22  }
23  weak relationship contains {
24      Course[1] -> Lecture[N]
25  }
26  relationship takes_exam {
27      Course[N] -> Professor[1]
28          -> Student[N]
29      points: double
30  }
31
32
```

```
CREATE TABLE Person (
    full_name varchar(255) NOT NULL,
    birthday datetime NOT NULL

    PRIMARY KEY (full_name)
);

CREATE TABLE Student (
    undergraduate bit NOT NULL,
    matr_nr int NOT NULL
    full_name varchar(255)

    PRIMARY KEY (matr_nr)
    FOREIGN KEY (full_name)
        REFERENCES Person(full_name)
);

CREATE TABLE Course (
    title varchar(255) NOT NULL,
    course_id int NOT NULL

    PRIMARY KEY (course_id)
);

CREATE TABLE Professor (
    pers_nr int NOT NULL
    full_name varchar(255)

    PRIMARY KEY (pers_nr)
    FOREIGN KEY (full_name)
        REFERENCES Person(full_name)
);
```

Fig. 2. An example Entity Relationship model (center) with excerpts of the textual concrete syntax (left) and the generated SQL code (right) in BIGER

ScopeProvider are replaced. With the use of Xtend[5], a code generator for an SQL schema is implemented.

The language and its language server are built and bundled into a single distribution, making it available as a CLI tool for the VS Code language client. Furthermore, a TextMate grammar is added to the extension for syntax highlighting. Besides Xtext, the BIGER tool also takes advantage of the Sprotty framework[6] for its versatile diagram modeling features. Adding a Diagram Generator with additional glue code provided through sprotty-xtext, maps the textual model elements to corresponding graphical representations. Sprotty also enables the use of the Eclipse Layout Kernel (ELK) to add auto-layout the ER model on the server. The graphical model is then sent to the client and with additional glue code provided by sprotty-vscode, the extension renders the diagram.

The next step is to display the diagram within the webview. Using the client part of Sprotty, the graphical model is passed to the webview and rendered into style-able SVG views. Such SVG rendered models are highly scalable, which is one of the shortcomings of many existing modeling tools when handling large models. Sprotty is based on actions and commands to add interactivity to diagrams and even extend the LSP to tunnel Sprotty Actions. BIGER takes advantage of this approach and thereby enables the final hybrid modeling support.

III. Using the BIGER Tool

To showcase the key features of the BIGER Tool we provide an example, which models a basic university environment and contains different ER concepts. Figure 2 shows the full example with the textual editor (left), the diagram (center), and the generated SQL code (right) opened in VS Code with the dark theme being activated.

To define a new model, we use the `erdiagram` keyword followed by an appropriate name at the top (line 1). Next we include the optional `generateSql` flag (line 2) to enable the SQL code generation and then start with the specification of the actual model elements. Students take exams of courses, which are held by one professor, so we specify a ternary relationship between those entities (line 26). Points are also saved for exams. They become visible in the diagram when hovering over the takes_exam relationship. The entities professor and student are defined by means of an inheritance (lines 9 and 13) to the generic person entity as both share the common attributes name, birthday, and age. Within the person entity the name acts as the primary key and since the age can be computed from the birthday, the attribute is marked as derived (line 7). Lectures are part of courses, however, they can not be identified on their own, so we specify lecture as a weak entity, provide a partial key and declare the corresponding relationship as weak as well (lines 20 and 23). Professors can publish publications and publications can be written by professors, which is modeled as a binary many-to-many relationship.

IV. Related Work

A wide variety of tools are available for ER modeling. However, the majority of them are more general database applications, disregarding the semantics of the underlying ER model and are solely intended for graphical representations, e.g. diagrams.net[7]. The focus of this section is on a brief comparison of related ER modeling tools.

[5]https://www.eclipse.org/xtend/
[6]https://projects.eclipse.org/projects/ecd.sprotty
[7]https://www.diagrams.net/

TABLE II
RELATED ER MODELING TOOLS AND VS CODE EXTENSIONS FOR ER

	DBDiagram	**ERD Plus**	**EERDSL**	**ERText**	**ERD Editor**	**ERD Preview**	**BIGER**
Web-based	✓	✓			✓		✓
IDE Integration			Eclipse	Eclipse	VS Code, Atom	VS Code	VS Code
Open Source			✓	✓	✓	✓	✓
Textual Editor	✓		✓	✓	✓	✓	✓
LSP Implementation	✓		✓	✓	✓	✓	✓
Diagram View	✓	✓	✓		✓	✓	✓
SQL Generation	✓	✓	✓		✓		✓
Model Validation	✓		✓	✓			✓
Interactive Diagram	✓	✓					✓

In the context of VS Code, there are two relevant extensions available, ERD Editor[8] with a focus on graphical modeling and ERD Preview[9] with a more basic diagram view, but offering textual modeling. Both extensions support generating SQL code but lack an interactive diagram.

When looking at web applications, DBDiagram[10] offers the Database Markup Language (DBML) to define database schemas, together with graphical modeling. ERD Plus[11] does not contain a textual language, but also allows the creation of ER diagrams by a purely graphical approach and additional transformations to Relational Schemas, Star Schemas, and SQL DDL statements.

Apart from web-based tools, the domain-specific language ERText allows specifying conceptual ER models and transforming them into logical ones [9]. For EER models the Multi-Paradigm Information System Modeling Tool with the EERDSL uses a bidirectional (graphical and textual) approach for conceptual modeling and supports transformations into a relational data model, or a class model [10].

Table II provides a compact comparison of various related approaches. It shows that many features have been realized in other tools, however, BIGER remains the only tool that provides an interactive diagram. Moreover, we can state that BIGER is the first real modeling tool realized following the Language Server Protocol and being distributed through the VS Code ecosystem. Being based on web technologies, BIGER can also easily be embedded in conventional web pages.

V. CONCLUSION AND FUTURE WORK

In this paper, we presented the BIGER Entity Relationship modeling tool. The tool is one of the first in its kind by realizing a language server and being distributed via the VS Code marketplace[3]. We released version 0.0.2 of BIGER together with the publication of this paper while already working on the next releases.

We are proposing the tool to colleagues in our networks responsible for teaching data modeling courses, hoping they would add BIGER amongst the tools they recommend for their course. Future versions of the tool will provide a richer set of functionality, e.g., the generation of different SQL dialects to ease the use of the code with different relational database management systems. Moreover, we are currently integrating advanced layouting algorithms to improve the rendering of the models, particularly the routing of the relations. From a research perspective, we also see potential of using BIGER to investigate the perceived usefulness and ease of use of graphical and textual modeling editors. A video showcasing BIGER in use can be found at: https://youtu.be/0jw05xChp7I.

[8]https://github.com/vuerd/
[9]https://github.com/kaishuu0123/vscode-erd
[10]https://dbdiagram.io/
[11]https://erdplus.com/

REFERENCES

[1] P. P.-S. Chen, "The entity-relationship model—toward a unified view of data," *ACM Trans. Database Syst.*, vol. 1, no. 1, p. 9–36, Mar. 1976.

[2] B. Thalheim, "Extended entity-relationship model." *Encyclopedia of Database Systems*, vol. 1, pp. 1083–1091, 2009.

[3] D. Bork, A. Garmendia, and M. Wimmer, "Towards a multi-objective modularization approach for entity-relationship models," in *ER Forum, Demo and Poster 2020*, J. Michael and V. Torres, Eds. CEUR, 2020, pp. 45–58. [Online]. Available: http://ceur-ws.org/Vol-2716/paper4.pdf

[4] R. Rodríguez-Echeverría, J. L. C. Izquierdo, M. Wimmer, and J. Cabot, "Towards a language server protocol infrastructure for graphical modeling," in *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2018, pp. 370–380.

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

[6] J. Cooper and D. Kolovos, "Engineering hybrid graphical-textual languages with sirius and xtext: Requirements and challenges," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion*. IEEE, 2019, pp. 322–325.

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

[8] A. Abbad Andaloussi, J. Buch-Lorentsen, H. A. López, T. Slaats, and B. Weber, "Exploring the modeling of declarative processes using a hybrid approach," in *Conceptual Modeling*, A. H. F. Laender, B. Pernici, E.-P. Lim, and J. P. M. de Oliveira, Eds. Cham: Springer International Publishing, 2019, pp. 162–170.

[9] J. Lopes, M. Bernardino, F. P. Basso, and E. d. M. Rodrigues, "Textual approach for designing database conceptual models: A focus group." in *MODELSWARD*, 2021, pp. 171–178.

[10] M. Čeliković, V. Dimitrieski, S. Aleksić, S. Ristić, and I. Luković, "A dsl for eer data model specification," 2014.