BRModelo Web Extension: Implementing a Textual Domain-Specific Language for Conceptual Database Modeling

Igor Andrey Ronsoni[1], **Geomar A. Schreiner**[1], **Denio Duarte**[1]

[1]Universidade Federal da Fronteira Sul (UFFS)
Chapecó – SC – Brazil

{igor.ronsoni,gschreiner,duarte}@uffs.edu.br

Abstract. *Data modeling is a fundamental step in the development of systems, as it allows domain requirements to be represented in a clear and structured manner. In the Brazilian BRModelo stands out as a widely adopted tool for creating Entity–Relationship (ER) diagrams, offering an visual interface. However, its limitation to exclusively graphical manipulation can reduce productivity and flexibility of use. This work proposes ConceptText, an extension of BRModelo WEB (BRMW) through the integration of a command-line (CMD) module with support for a dedicated textual language for the creation and manipulation of conceptual diagrams. The solution was evaluated through user experimentation, and the results indicated good acceptance, preservation of the original usability, and a perceived increase in efficiency in the modeling process. This approach contributes to its modernization and integration with contemporary development workflows.*

1. Introduction

Data modeling is one of the fundamental phase in the development of information systems, as it enables the abstract representation of data and their interrelationships within a given organizational or business context [Belcic and Stryker 2024]. This activity goes beyond mere documentation of requirements. Several factors, including differing interpretations of the application domain and design decisions, contribute to the existence of multiple equally valid models for the same real-world scenario [Simsion and Witt 2005].

Among the various approaches to data modeling, conceptual modeling captures data requirements in a manner independent of specific implementation technologies [Simsion and Witt 2005]. The Entity–Relationship (ER) model has been used for represent data objects and their relationships in the conceptual abstraction level [Simsion and Witt 2005]. Over time, data modeling tools have evolved from strictly visual solutions to more sophisticated environments that incorporate textual manipulation, collaborative features, and version control.

Modern tools such as dbdiagram.io[1] and QuickDBD[2] support diagram creation and editing via simplified textual languages. These platforms adopt an approach that describes entities, attributes, and relationships through structured commands. This strategy tends to enhance productivity and mitigate the common ambiguities inherent in purely

[1]https://dbdiagram.io/home
[2]https://www.quickdatabasediagrams.com/

graphical editors [Lopes et al. 2021]. Despite these advances, such tools are predominantly oriented toward logical data modeling. Furthermore, the related work survey revealed that although textual languages for ER modeling exist, few provide robust support for constructs such as specializations and associations, which are frequently required for modeling more complex conceptual schemas.

Accordingly, this work proposes the design of a dedicated Domain Specific Language (DSL) specifically oriented toward conceptual modeling based on the Extended Entity–Relationship (EER) model, called *ConceptText*[3] [4]. The language was defined to encompass the primary constructs used in BRModelo Web (BRMW)[Neto et al. 2016], thereby enabling users to specify complete models using clear, concise textual commands.

This integration yields a hybrid textual–visual approach that expands the range of modeling possibilities and aligns with current trends in software engineering tools. The implemented solution enables models to be specified, validated, and automatically transformed into diagrams, while preserving the tool's original usability. We conduct a set of experiments to evaluate the effectiveness of *ConceptText* regarding: usability, clarity, and seamless integration with the brModelo graphical environment. The results show that *ConceptText* is a relevant improvement for agile conceptual modeling. Additionally, all participants agreed that the DSL is clear and supports rapid learning.

2. Related Work

Several tools address textual data modeling predominantly at the logical schema level, such as *dbdiagram.io* and *QuickDBD*. These tools employ lightweight Domain-Specific Languages (DSLs) that facilitate rapid schema specification but do not provide support for Extended Entity–Relationship (EER) conceptual constructs, including specialization/generalization hierarchies and weak entities. Within the broader landscape of formal DSLs for graphical editors, *Xdiagram* [Santos and Gomes 2016] generates visual interfaces from metamodel specifications, whereas *HyLiMo* [Krieger et al. 2024] extends this paradigm by offering a fully synchronized hybrid text–visual modeling environment. Focusing specifically on conceptual modeling, *MIST* [Dimitrieski et al. 2014] employs *EERDSL* to provide expressive conceptual abstractions, but does not support a hybrid text–diagram workflow. In contrast, *ERtext* [Lopes et al. 2021] offers a textual interface explicitly designed to unify conceptual, logical, and physical modeling phases. Diverging from ERtext, our approach deliberately limits its semantic expressiveness to the EER conceptual level, thereby ensuring direct compatibility with the BRModelo Web ecosystem rather than pursuing the unification of multiple abstraction tiers.

ConceptText differs from the related work as the follows: (*i*) tackling the conceptual model, (*ii*) proposing a dedicated DSL, (*iii*) offering an integrated visual editor, (*iv*) supporting ER/EER constructs, and (*v*) the capability for real-time bidirectional synchronization between textual and graphical representations.

[3]Github project: `https://github.com/Igorronsoni/brmodelo-app`
[4]Demonstration video: `https://tinyurl.com/3phfzcnp`

3. ConceptText Development

ConceptText is an independent, decoupled text editor module within the *BRMW*[5]. This module provides a dedicated environment for textual Extended Entity-Relationship (EER) modeling using a custom Domain-Specific Language (DSL). The editing infrastructure is powered by *CodeMirror*, configured to provide real-time syntax highlighting and token formatting. The Figure 1 presents a overview of the interface of BrModelo Web, in the left (a) is the text editor where the commands can be added (*ConceptText*) and in the right (b) the graphical area to view and edit the model (original BRMW interface).

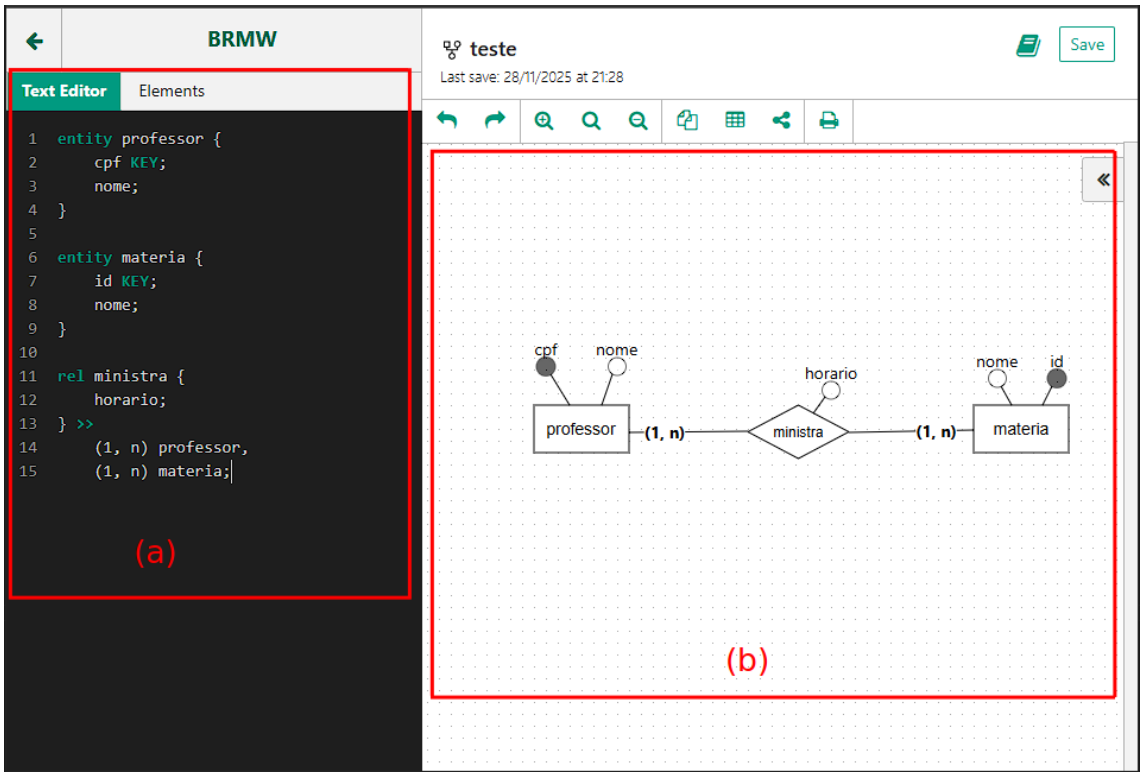

Figure 1. BrModelo Web GUI

The system architecture is built on a parser implemented with Nearley.js, using a Context-Free Grammar (CFG) defined in EBNF to validate DSL commands. The parsing process generates an Abstract Syntax Tree (AST) that hierarchically describes the model. A transformation layer then converts this AST into an optimized JSON format. This strategy enables the diagram engine to perform differential rendering by comparing the current graph with the new state, mitigating performance bottlenecks and ensuring fluidity during the generation of visual EER diagram elements.

The Text Editor Module was designed with the principle of decoupling, functioning as an independent, reusable extension within the *BRMW* ecosystem. The interface leverages the *CodeMirror* library, configured with dynamic tokenization rules to provide real-time syntax highlighting and lexical support.

A set of fundamental callback functions orchestrates the synchronization between the textual and graphical interfaces: (*i*) Syntax and Interpretation: Apply grammatical

[5]https://github.com/Igorronsoni/brmodelo-app

rules and perform semantic analysis (e.g., type checking and duplicate detection), (*ii*) Transformation and Update: Convert the AST to JSON and manage persistence and automatic saving of the model state, and (*iii*) Initialization: Ensures the reconstruction of the textual environment from saved metadata.

This structure establishes a bidirectional hybrid modeling environment in which the textual specification and the graphical representation of the conceptual model coexist with full integrity and synchronicity.

3.1. Lexical and Syntactic Specification

The developed DSL formalizes the textual notation for conceptual modeling proposed by Heuser (1998). The lexical analyzer, implemented via *moo.js*, processes characters into meaningful tokens such as keywords (`entity`, `rel`, `specialize`), *identifiers*, *delimiters*, and cardinality constraints.

A Context-Free Grammar (CFG) parser was implemented using the *Nearley.js* library. Below, we present the EBNF representation of the grammar utilized by the interpreter to validate user input and construct the AST:

```
<main> ::= <declaration> ( <declaration> | <null> )*
<declaration> ::= <entity_nd>
                | <assentity_nd>
                | <rel_nd>
                | <note_nd>
                | <specialize_nd>
<entity_nd> ::= ENTITY  <identifier>  {<attributes>}
<attributes> ::= <attribute> {<attribute>}
<attribute> ::= (<cardinality> <identifier>;)
              | (<identifier> KEY;)
              | (<identifier> COMPOSED [<attributes_composed>])
<identifier> ::= IDENTIFIER | STRING
<cardinality> ::= (<zero_or_one> , <one_or_n>)
<attributes_composed> ::= <attribute_composed>
    { <attribute_composed> }
<attribute_composed> ::= <cardinality> <identifier> ;
<specialize_nd> ::= SPECIALIZE {<specialize_types>}
    <specialize_entity_ref> <specialize_entity_list>;
<specialize_types> ::= ( <t_or_p> , <d_or_c> )
<t_or_p> ::= IDENTIFIER
<d_or_c> ::= IDENTIFIER
<specialize_entity_list> ::= <specialize_entity_ref>
    ( <specialize_entity_ref> )*
<specialize_entity_ref> ::= IDENTIFIER
<rel_nd> ::= REL [<identifier>] <rel_attributes> <rel_entities>;
<rel_attributes> ::= {<attributes>}
<rel_entities> ::= <rel_entity_ref> (<rel_entity_ref> )*
<rel_entity_ref> ::= [<cardinality>] <identifier> [WEAK] [STRING]
<zero_or_one> ::= 0 | 1
<one_or_n> ::= 1 | N
```

The syntactic verification process is initiated at the root <main> production, which evaluates a <declaration> that resolves to an <entity_nd> (entity node). In this phase, the parser consumes the terminal keyword 'ENTITY', retrieves the entity identifier via the <identifier> non-terminal, and processes the enclosed block of properties specified by {<attributes>}. Within this scope, the <attributes> production prescribes a sequence of one or more <attribute> instances. For simple entities, these individual attributes are restricted to primitive structural properties, such as primary identifiers (<identifier> KEY;) or conventional attributes associated with explicit participation constraints (<cardinality> <identifier>;). This formal derivation procedure incrementally decomposes the textual specification, thereby ensuring that both the entity's external interface (signature) and its internal attribute set are rigorously verified and systematically mapped onto the AST representation required for subsequent diagram generation.

3.2. Semantic Analysis and Diagram Generation

The parsing process creates an AST, which immediately undergoes semantic analysis. This verification layer ensures model integrity by detecting duplicate names, validating attribute types, and checking the structural consistency of relationships and specializations. Real-time feedback is provided to the user directly in the editor.

Once syntactically and semantically validated, a transformation class converts the AST into a standardized *JSON* format. This intermediate JSON acts as the bridge to the graphical module, feeding directly into the *JointJS* rendering engine used by *BRModelo Web*. For example, a simple entity like `entity Dependent {name}` is dynamically translated into JSON and subsequently rendered as an entity box on the canvas. This pipeline establishes a fully bidirectional, hybrid modeling environment where text and visual diagrams are perfectly synchronized and continuously persistable.

4. Experiments

To evaluate user acceptance of *ConceptText* and its integration into BRMW, an online questionnaire using a *Likert scale* was applied to students enrolled in an Advanced Databases course, yielding 10 responses. The instrument evaluated several dimensions of the solution, including the perceived usefulness of the textual approach, clarity of the DSL, ease of learning, and the overall impact on usability. These aspects were examined through a combination of quantitative (closed-ended) items and qualitative (open-ended) questions.

The responses show a high rate of user approval: 70% of respondents rated the proposal as *Very Useful* and the remaining 30% evaluated it as *Useful*. This distribution underscores the textual approach as a relevant improvement for agile conceptual modeling. Additionally, nine out of ten participants agreed or strongly agreed that the DSL is clear and supports rapid learning. Nonetheless, the qualitative responses suggested that extending the documentation and including more illustrative examples could further improve the learning experience. Importantly, all respondents reported that the original usability of BRMW was preserved, suggesting that the DSL provides a faster, more expressive modeling interface without disrupting the existing workflow.

The open-ended comments reinforced the quantitative results, frequently describing the DSL as an intuitive and efficient mechanism for schema specification. Participants

especially highlighted that the tight coupling between textual specifications and visual diagrams enhanced their sense of control over the database design process. Taken together, these findings support the effectiveness of the proposed hybrid approach and indicate a positive impact on the overall user experience in conceptual data modeling.

5. Conclusion

This work presented *ConceptText*, an approach to integrate a DSL into the BRModelo Web environment, enabling text-based specification and manipulation of Entity-Relationship (ER) diagrams. By combining a dedicated textual editor with a visual modeling interface, we define a hybrid modeling workflow that supports both declarative specification and graphical exploration of conceptual schemas. The DSL is defined by a formal grammar and processed through lexical, syntactic, and semantic analyses, which collectively reduce manual modeling errors and increase structural accuracy and consistency.

The proposed approach was empirically evaluated for performance and usability. The results indicate that complete conceptual schemas can be specified purely in textual form and automatically synchronized with their corresponding visual representations in real time. Furthermore, the parser's built-in validation mechanisms systematically prevent the introduction of invalid constructs, thereby demonstrating that bidirectional text–visual modeling can lead to a more efficient, reliable, and robust conceptual design process.

References

Belcic, I. and Stryker, C. (2024). O que é diagrama de entidade e relacionamento (erd)? `https://www.ibm.com/br-pt/think/topics/entity-relationship-diagram`. Abr. 2025.

Dimitrieski, V., Čeliković, M., Aleksić, S., Ristić, S., and Luković, I. (2014). Extended entity-relationship approach in a multi-paradigm information system modeling tool. In *2014 FedCSIS*, pages 1611–1620.

Krieger, N., Speth, S., and Becker, S. (2024). HyLiMo: A hybrid live-synchronized modular diagramming editor as ide extension for technical and scientific publications. In *1st ACM/IEEE Workshop on Integrated Development Environments*.

Lopes, J., Bernardino, M., Basso, F., and Rodrigues, E. (2021). Textual approach for designing database conceptual models: A focus group. In *MODELSWARD*, pages 171–178.

Neto, M. B. d. S. et al. (2016). brmodeloweb: Ferramenta web para ensino e modelagem de banco de dados.

Santos, A. L. and Gomes, E. (2016). Xdiagram: a declarative textual dsl for describing diagram editors (tool demo). In *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*, pages 253–257.

Simsion, G. and Witt, G. (2005). *Data Modeling Essentials*. Morgan Kaufmann, 3 edition.