Human Systems Engineering and Design (IHSED2023), Vol. 112, 2023, 436–445
https://doi.org/10.54941/ahfe1004156

Modeling Low-Code Databases With Executable UML

Alan Bubalo and Nikola Tanković

Faculty of Informatics, Juraj Dobrila University of Pula, Zagrebačka ulica 30, 52100 Pula, Croatia

ABSTRACT

This study aims to create a method for transforming a Unified Modeling Language (UML) class model into an open-source end-user database. The manual transformation of UML class models into a database can be time-consuming and prone to errors. By making a database schema from a UML class model in standard XMI format, our framework offers an automated alternative and makes the transformation more useful. The tool gets the tables, attributes, and connections that compose the schema from the class model's classes, properties, and relationships. The tool also has an abstract RESTful web service component to give the newly made database a web interface. Such a tool will make it easier for software engineers with less experience, especially students, to learn and use UML class models. We implemented the framework for a Baserow end-user database and evaluated it on a student internship use case. The accompanying code is available as an open-source GitHub repository.

Keywords: UML class diagram, Model-driven development, Low-code database, Baserow, REST API

INTRODUCTION

Background and Motivation

Due to the rapid development of technology, databases are now a necessary part of most software programs. These databases can be time-consuming, difficult to maintain, and prone to mistakes, especially when design models are converted into a database schema. As a result, there is an increasing demand for straightforward and practical solutions to speed up this process. Businesses seek solutions to meet their software needs quickly, cheaply, and securely. Low-code development platforms (LCDPs) have come to light in this context as promising strategy to assist businesses in achieving these objectives (Talesra and Nagaraja, 2021). It is why low-code databases, which reduce human coding and streamline development, are gaining popularity.

Software engineers use the widely recognized modeling language, UML, to visualize and express the design of a software system. UML class models typically represent database schemas, making the conversion between the two a crucial stage in development. The Unified Modeling Language class diagram (UML) has evolved into an industry standard for modeling time-varying data

from the real world into objects (Soumiya and Mohamed, 2017), emphasizing the importance of converting UML class models into database schemas. However, manually translating UML class models into a database design can be time-consuming and error-prone, highlighting the need for efficient and reliable conversion methods.

This project attempts to develop a technique for converting a UML class model into an open-source end-user database in response to these difficulties. Our system decreases the time and errors involved with manual conversion by automating the translation process and providing a RESTful web service component. For students and less experienced software developers, such a tool offers great promise for speeding up the study of and use of UML class models.

Objectives

The primary objectives of this research are:

- Create a framework that automatically converts UML class models into an open-source end-user database schema.
- To implement an abstract RESTful web service component that offers a web interface for the freshly made database.
- Evaluate the framework's performance, usefulness, and application in a student internship case study.

Scope and Limitations

The main objectives of this study are constructing and assessing a framework for converting UML class models into an open-source end-user database, explicitly using the Baserow end-user database. The study needs to cover various modeling languages and various varieties of UML diagrams.

Although the suggested framework provides an automatic substitute for manual conversion, there might still be restrictions on how it can be used in more complicated or specialized situations. Additionally, the rating is based on a use case from a student internship, which may not fully represent all scenarios that occur in real-world software development environments. The study, however, lays the groundwork for future research and advancements in model-driven engineering, which aims to boost overall quality and automate various software development processes to maximize efficiency, as well as in low-code databases (Hutchinson et al., 2014).

LITERATURE REVIEW

In a study published in 2014, Li et al. (2014) looked at the conversion of UML class diagrams, a component of the Platform-Independent Model (PIM), into HBase. This NoSQL database is a part of the Platform-Specific Model (PSM). Their research compared the meta-models of UML class diagrams and HBase, mapped their respective feature sets, and developed mapping criteria. Several restrictions were placed on the research, such as potential model development flaws. Another restriction was the absence of debugging capabilities. Furthermore, the study should have examined the performance of more sophisticated models.

Byrne and Shahzad Qureshi (2013) explain the differences between using ER models and UML class diagrams. They conclude that while UML works well for abstract modeling, the IDEF1X approach is preferable for database design. ER models perform better than UML class diagrams during the requirements analysis process. As Byrne and Shahzad Qureshi indicate, UML class diagrams are more of a tool for developers than for database administrators. Additionally, they contend that because UML class diagrams are feature-rich, they are inappropriate for conceptual modeling. They did not investigate the differences between many-to-many relationships, ternary relationships, and aggregation in ER models and UML class diagrams.

In order to include the learning capabilities of UML class diagrams in ACME, Soler et al. (2010) analyze the distinctions between ER modeling and UML class diagrams. At the authors' university, an ACME web environment supports learning across various courses. Students can practice class diagram material using a web-based tool by modeling a class diagram following the specifications of the tasks in the exercises. The tool also evaluates the solution by associating parameters or attributes from the UML class diagram with the student's chosen solution. In an experiment, they demonstrated that students who utilized the program outperformed students who completed the assignments manually. A more significant evaluation involving more pupils needs to support the conclusions.

Similar to Li, Gu, and Zhang, Feng et al. (2015) describes how to use annotations to convert UML class diagrams into Cassandra (NoSQL) data models. The authors map the meta-models of Cassandra models to UML class diagrams. Before the transformation, users must announce or clarify which properties correspond to which sort of key (partition, composite key, and clustering key). The number of collections in the modified Cassandra model exceeds the number of classes in the UML class diagram. Because users must manually add annotations, the procedure is not automated.

To assist students in their study of model-driven engineering (MDE), Batory and Azanza (2017) created a tool called MDELite. The meta-model of a relational database was specified using the Prolog programming language. They also used translation techniques such as Model-to-Text (M2T) and Tool-to-Model (T2M). The authors did not present any data to support their claim that MDELite increases students' productivity in mastering MDE.

Brdjanin and Maric (2012) offer a method for automatically developing an initial conceptual database model using UML activity diagrams frequently used to represent business operations. They create a set of formal rules that may extract participants and business objects from activity diagrams and produce corresponding classes and associations in a UML class diagram representing the conceptual database model. Using a real-world business model, the authors constructed an automatic generator based on these guidelines and tested its effectiveness. There is still opportunity for further research and advancements because the study needs to say more about the effects of control patterns or the viability of expanding the rules to cover entire company models.

METHODOLOGY

Specifically focusing on the Baserow end-user database, this section describes the process used to construct the framework for converting UML class models into an open-source end-user database and developing an abstract RESTful web service component using Python's Flask. The methodology has four main steps: Using StarUML, create the UML class model. Then, extract the pertinent data from the produced XMI file. Then, use Baserow to create the database schema. Finally, use Flask to develop the REST API. The structure of using the application is shown below (Fig. 1).

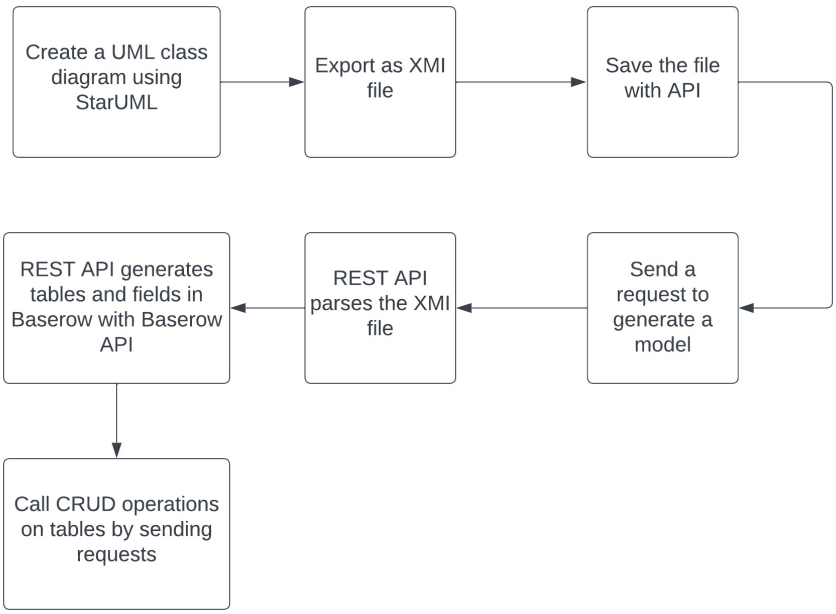

Figure 1: Structure of using the API.

The suggested framework is made to automatically turn UML class models into Baserow database schemas and provide a RESTful web interface for interacting with the resulting databases. The following workflow forms the foundation of the framework:

- Use the StarUML program to create a UML class model.
- Using the XMI extension, export the model as an XMI file.
- Use the created program to process the XMI file and extract the classes, attributes, and associations.
- Send Baserow API calls to build tables and fields based on the extracted data.
- Use Flask (Python) to implement a RESTful API to give the newly constructed database a web interface.

With the help of the StarUML program, a UML class model is first created, allowing for the visual representation of the database schema, including the

classes, attributes, and associations. The class model should appropriately reflect the intended structure and relationships of the database.

The StarUML-provided XMI extension exports the finished UML class model as an XMI file. The information required to create the database schema in Baserow is in the XMI file, which is used as the application's input.

The program extracts classes, attributes, and associations after it has read the XMI file. While modeling, users need to add annotations to the attributes of a class to specify the corresponding field types in Baserow, such as adding a "created on" annotation to create a "Created On" field type in Baserow. With the help of this data, the application asks the Baserow API to add the necessary tables and columns to the Baserow end-user database. The relationships, structures, and annotated field types specified in the original UML class model are reflected in the created database schema in Baserow.

Finally, a web interface for Baserow's freshly built database is implemented using Flask (Python) and a RESTful API. To efficiently interact with the database, the API includes a variety of query parameters and filtering options in addition to conventional CRUD operations (Create, Read, Update, Delete). Developers can use the API to include the created database in their applications, providing a simple and effective method for managing and modifying data.

IMPLEMENTATION

This section explains how the suggested architecture is implemented to convert UML class models into an open-source end-user database using the Baserow platform and create a RESTful API using Flask (Python). The four main steps in the implementation process are as follows: The creation of the tool, the extraction of tables, attributes, and connections from the XMI file, the generation of the database schema in Baserow, and the use of Flask to construct the RESTful web service component are all steps in the process.

Design and Development of the Tool

The created program is intended to be used as a command-line tool that takes an input XMI file and constructs the appropriate Baserow database schema. Python was used to create the tool because of its vast library support and user-friendly interface. The application comprises various modules that interface with the Baserow API, parse the XMI file, extract the pertinent data, and construct the RESTful API using Flask.

The app provides the following functionalities:

- Uploading, updating, and deleting XMI files through the "/files/" route.
- Performing CRUD operations on UML models through the "/uml_models/" route, which includes providing Baserow database credentials, generating the database based on the uploaded file, and managing the uploaded models.
- Mapping components retrieved from the XMI reader to Baserow tables and fields using the *baserow_init* module.
- Sending requests to Baserow through the *baserow_client* module.

- Implementing CRUD operations for various routes using the API files within the "*app/api*" folder.

The app allows users to upload XMI files and interact with them using the *"/files/"* and *"/uml_models/"* routes. The "*baserow_client*" module communicates with Baserow, while the *baserow_init* module processes the XMI files, filters components, and maps them to Baserow tables and fields. The API files within the *app/api* folder handle the CRUD operations for the different routes, enabling users to interact with the data and the generated databases.

Extraction of Tables, Attributes, and Connections From the XMI File

The application reads the XMI file and uses an XML parser to extract the necessary data from the UML class model regarding classes, attributes, and associations. The parser finds classes, attributes, and associations by iterating over the XML elements and filtering according to the proper tags and attributes. After the pertinent data has been collected, it is saved in data structures (such as dictionaries and lists) to be processed later.

Generating the Database Schema in Baserow

The application sends requests to the Baserow API for the Baserow end-user database's corresponding tables and fields to be created using the data extracted from the XMI file. Table creation, field adding, and establishing connections between tables are all requests sent through the API. The program also manages any faults and exceptions, such as invalid input, login problems, and server errors, that may arise while communicating with the Baserow API.

RESTful Web Service Component Implementation With Flask

A RESTful API uses Flask to give a web interface for interacting with the newly created database after the database schema has been generated in Baserow. In addition to additional data querying and filtering endpoints, the Flask application defines several routes for managing CRUD operations (Create, Read, Update, Delete) on the database tables.

In order to enable secure access to the database, the RESTful API implementation also contains mechanisms for user input validation, appropriate error handling, and authentication. Developers that want to incorporate the produced database into their applications can access the Flask application by deploying it on a web server.

In conclusion, implementing the suggested framework streamlines the development process and makes it more effective for software engineers, especially those with less experience or students learning UML class modeling. It automates the conversion of UML class models into an open-source end-user database in Baserow and offers a RESTful API using Flask.

The implementation of this project can be found in our GitHub repository https://github.com/AlanBubalo/modeling-low-code-databases-with-executable-uml.

EVALUATION

The efficiency of the suggested framework in converting UML class models into an open-source end-user database using Baserow and offering a RESTful API using Flask is the main emphasis of the evaluation. There are four primary components to the evaluation: (1) the student internship use case, (2) the efficiency of the transformation process, and (3) a comparison to other approaches.

Student Internship Use Case

A student internship use case, which consists of a UML class model depicting a database structure for managing internships, students, and related information, is chosen to evaluate the framework. The developed application is used to construct the UML class model using StarUML, export it as an XMI file, and convert it into a Baserow database schema. Flask is used to implement the RESTful API and provide a web interface for interacting with the created database.

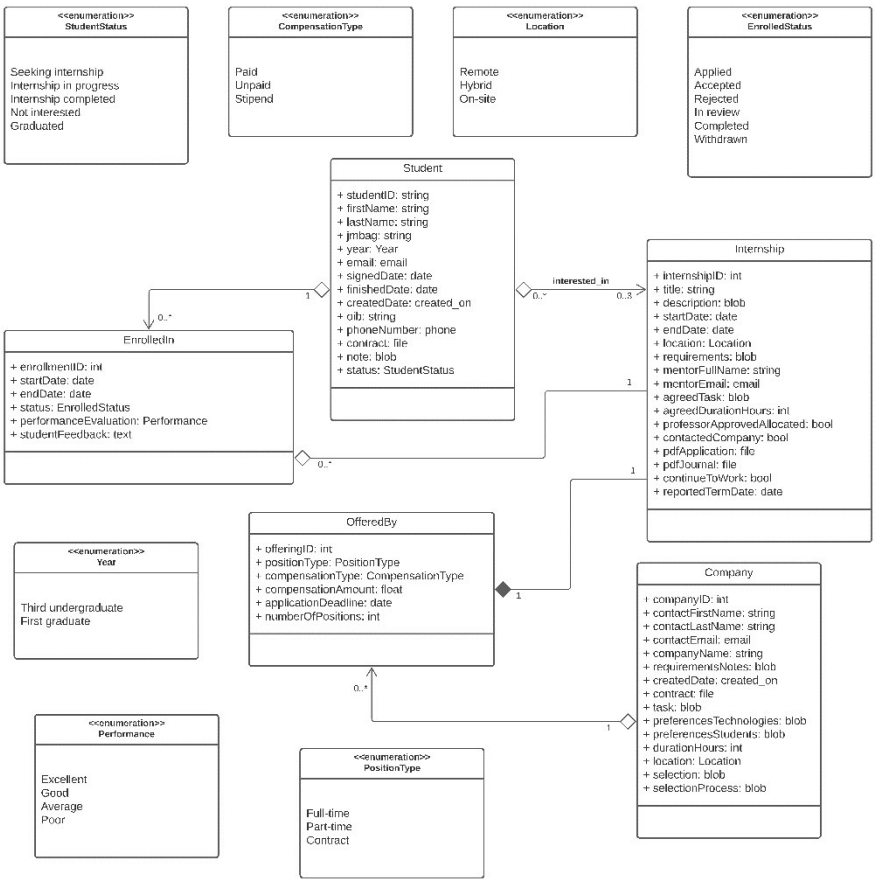

Figure 2: Student internship UML class diagram.

Effectiveness of the Transformation Process

The produced Baserow database schema is compared to the original UML class model to determine how well the transformation process worked. The evaluation focuses on handling mistakes and exceptions throughout the transformation process and the accuracy of table creation, field insertion, and establishing linkages across tables. The outcomes show that the suggested framework accurately converts the UML class model into a Baserow database schema, saving time and effort over manual conversion. The UML class diagram used to evaluate is shown in Fig. 2.

Comparison to Existing Solutions

The suggested framework is then contrasted with current approaches for creating RESTful APIs and converting UML class models into database schemas. The level of automation, usability, features supported, and overall performance are the main comparing criteria. The evaluation suggests that the framework is a valuable addition to the low-code databases and model-driven development fields since it provides a rare mix of automatic transformation, an abstract RESTful web service component, and a smooth connection with Baserow.

In conclusion, the evaluation shows how well the suggested framework for converting UML class models into an open-source end-user database in Baserow and offering a RESTful API using Flask works, as well as how usable it is and how it can be applied. The framework can speed up the development process and make it easier for students and less experienced software developers to learn and apply UML class modeling.

DISCUSSION

The main conclusions, restrictions, and implications of the suggested framework for converting UML class models into an open-source end-user database using Baserow and offering a RESTful API using Flask are discussed in the discussion section. Potential areas for further research are also noted.

Several important conclusions were made once the proposed framework was put into practice and evaluated, including:

- The framework efficiently automates the translation of UML class models into a Baserow database schema, avoiding potential errors and the time and effort needed for manual conversion.
- The user-friendly and intuitive framework makes it easier for students and less experienced software engineers to learn and apply UML class modeling.
- The RESTful API implementation using Flask provides a web interface for interacting with the generated database, improving its usability and accessibility in distributed environments.

Despite the encouraging outcomes, the suggested paradigm has certain drawbacks. The study focuses on Baserow as the intended end-user database. It may be necessary to make further adjustments or expansions to make the framework applicable to other databases and platforms.

The evaluation is based on a use case from a student internship, which could only include some scenarios that could occur in real-world environments for software development.

The suggested paradigm has some consequences for model-driven development and low-code databases. The framework can accelerate development and increase productivity by automating the conversion of UML class models into database schemas and offering a RESTful API. The framework can also be a helpful teaching tool for students and less experienced software engineers, assisting them in better comprehending and applying UML class modeling in practical situations.

CONCLUSION

In this study, we proposed a framework for converting UML class models into a free, open-source end-user database and a RESTful API using Baserow and Flask. The framework automates the conversion procedure, avoiding potential errors and the time and effort needed for human transformation. Based on a use case for a student internship, our assessment showed how practical, applicable, and compelling the suggested framework is, particularly for students and less-experienced software engineers.

Although the framework has some drawbacks, such as a single-use case evaluation and a focus on Baserow as the target database, it nonetheless significantly contributes to low-code databases and model-driven programming. It might speed up the development procedure, make learning and using UML class modeling more manageable, and increase overall output.

Future development might improve the RESTful API implementation, study performance, and scalability, and conduct more thorough user research. It could also be extended to support additional databases and platforms. The suggested framework could enhance the understanding and use of UML class modeling and advance the development process in many scenarios.

The project's GitHub repository offers an example folder with sample files and resources to help users understand and use the presented methodology.

REFERENCES

About the Unified Modeling Language Specification Version 2.5.1. https://www.omg.org/spec/UML/2.5.1/About-UML/

Batory, D., Azanza, M.: Teaching model-driven engineering from a relational database perspective. Softw. Syst. Model. 16, 443–467 (2017).

Brdjanin, D., Maric, S.: An approach to automated conceptual database de-sign based on the UML activity diagram. Comput. Sci. Inf. Syst. (21), 249–283 (2012).

Byrne, B., Shahzad Qureshi, Y.: The use of UML class diagrams to teach database modeling and database design. In: Procs of the 11th Int Workshop on the Teaching, Learning and Assessment of Databases (TLAD). The Higher Education Academy (2013).

Feng, W., Gu, P., Zhang, C., Zhou, K.: Transforming UML Class Diagram into Cassandra Data Model with Annotations. In: 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity), pp. 798–805. IEEE (2015).

Hutchinson, J., Whittle, J., Rouncefield, M.: Model-driven engineering prac-tices in industry: Social, organizational and managerial factors that lead to success or failure. Sci. Comput. Program. 89, 144–161 (2014).

Li, Y., Gu, P., Zhang, C.: Transforming UML class diagrams into HBase based on the meta-model. In: 2014 International Conference on Information Science, Electronics and Electrical Engineering, vol. 2, pp. 720–724. IEEE (2014).

Soler, J., Boada, I., Prados, F., Poch, J., Fabregat, R.: A web-based e-learning tool for UML class diagrams. In: IEEE EDUCON 2010 Conference, pp. 973–979. IEEE (2010).

Soumiya, A. E. H., Mohamed, B.: Converting UML class diagrams into tem-poral object relational database. Int. J. Electr. Comput. Eng. 7(5), 2823 (2017).

Talesra, K., Nagaraja, G. S.: Low-code platform for application development. Int. J. Appl. Eng. Res. 16(5), 346–351 (2021).