Journal of Theoretical and Applied Information Technology
15th August 2024. Vol.102. No. 15

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A MODEL-DRIVEN APPROACH TO TRANSFORM UML MODELS INTO MONGODB SCHEMAS USING QVTO: FROM PIM TO PSM

HAMZA NATEK[1], AZIZ SRAI[2], ABDELMOUNAIM BADAOUI[3], FATIMA GUEROAUTE[4]

[1,3,4]Laboratory LASTIMI, High School of Technology Sale, Mohammed V University, Rabat, MOROCCO

[2]ERCI2A, FSTH, Abdelmalek Essaadi University, Tetouan, MOROCCO

E-mail: [1]hamzanatek@gmail.com, [2]a.srai@uae.ac.ma, [3]abdelmounaim.badaoui@est.um5.ac.ma, [4]fatima.guerouate@est.um5.ac.ma

ABSTRACT

Translating UML models into efficient NoSQL databases is a complex task within the domain of software engineering. This study addresses the problem of transforming UML models into MongoDB collections using the Model-Driven Architecture (MDA) approach. The research method involves defining metamodels for UML and MongoDB, followed by the development of QVTo transformation scripts to map UML class diagrams to MongoDB document structures. The transformation process was tested on a sample UML model, ensuring the correctness of the generated MongoDB schema. The findings demonstrate that the QVTo transformation scripts can accurately convert UML models into MongoDB collections, preserving the integrity and semantics of the original UML diagrams. The conclusions highlight the effectiveness of the MDA approach in bridging the gap between UML-based designs and NoSQL database implementations, providing a robust solution for data management in complex systems.

Keywords: *Model-Driven Architecture (MDA), UML to NoSQL Transformation, QVTo Transformation Scripts, MongoDB Schema Generation, Automated Data Modeling*

1. INTRODUCTION

In the realm of software engineering, Model-Driven Architecture (MDA) has emerged as a pivotal framework for automating the transformation of high-level models into executable artifacts. This paradigm shift facilitates the design, analysis, and implementation of complex systems through systematic model transformations. Unified Modeling Language (UML) serves as a standard modeling language that enables the visualization, specification, and documentation of software system artifacts [1]. However, transitioning from UML models to NoSQL databases, particularly MongoDB, presents significant challenges due to the inherent differences between relational and NoSQL data structures. MongoDB, a prominent NoSQL database, offers flexible and scalable data management solutions, making it ideal for modern applications that require dynamic schema evolution. Despite its advantages, converting UML class diagrams into MongoDB's document-oriented structure remains a complex task that necessitates an automated and accurate transformation process. Existing literature highlights various model transformation techniques, yet there is a notable gap in effective methodologies for converting UML models to MongoDB schemas, which this study aims to address.

This study not only fills the gap in existing methodologies but also enhances IT knowledge by introducing novel insights into the automated transformation process. The proposed methodology leverages the strengths of MDA to automate the conversion of UML models into MongoDB schemas, providing best practices that go beyond incremental knowledge. These include the precise definition of metamodels for both UML and MongoDB, the development of robust QVTo transformation scripts, and the validation of the transformation process to ensure the generated MongoDB schemas maintain the integrity and semantics of the original UML models. By synthesizing and integrating information from various sources, this research offers a comprehensive approach that enhances the efficiency and accuracy of database schema generation, contributing significantly to the field of model-driven engineering.

The primary problem addressed in this study is the lack of automated and accurate methods for transforming UML models into MongoDB schemas while preserving the semantic integrity and structure of the original UML diagrams. To tackle this problem, the study evaluates the practical benefits of integrating MDA with NoSQL databases, determining improvements in database management efficiency through comparisons with case studies and performance metrics from existing literature. It explores the methodological aspects of implementing MDA methodologies using JavaScript, reviewing best practices and validating guidelines for broader adoption. Additionally, the study assesses the theoretical and practical contributions to the broader field of IT knowledge by synthesizing prior knowledge and new insights. The validity of the study is ensured through rigorous testing and validation of the transformation process, confirming that the generated MongoDB schemas accurately reflect the original UML models. Peer reviews and expert consultations further substantiate the findings. Lastly, the study examines the incremental value of the research by comparing it with previous studies, determining whether the new methodology offers significant improvements in terms of efficiency, accuracy, and applicability.

The main objectives of this research are to develop a comprehensive methodology for transforming UML models into MongoDB collections using QVTo transformation scripts, define and implement precise metamodels for both UML and MongoDB to facilitate accurate mappings, and validate the transformation process to ensure the generated MongoDB schemas maintain the integrity and semantics of the original UML models. The scope of this study includes the development and testing of QVTo transformation scripts to map UML class diagrams to MongoDB document structures. This study does not cover the transformation of other UML diagram types or the integration with other NoSQL databases.

This study employs the Model-Driven Architecture (MDA) framework to guide the transformation process. We define detailed metamodels for UML and MongoDB and utilize QVTo transformation scripts to automate the mapping of UML class elements to MongoDB document structures. The transformation process is validated through rigorous testing on sample UML models, ensuring the accuracy and consistency of the generated MongoDB schemas.

The findings of this research hold significant potential for improving the efficiency and accuracy of database schema generation from UML models. By providing a robust and automated solution for transforming UML models into MongoDB schemas, this study contributes to the existing body of knowledge in model-driven engineering and offers practical implications for developers and engineers. The methodology developed herein can be adapted for other NoSQL databases, further extending its impact and applicability.

The structure of this paper is as follows: the Introduction provides context, problem statement, objectives, approach, significance, and an overview of the paper. The Related Work section reviews existing literature on model transformation techniques and identifies gaps that this study addresses. The Methodology section describes the MDA framework, metamodel definitions, and QVTo transformation scripts in detail. The Implementation section details the environment setup, transformation execution, and validation process. The Case Study section presents a sample UML model and demonstrates the transformation process and results. The Results and Discussion section discusses the findings, performance evaluation, challenges, and limitations. The Conclusion section summarizes the main contributions, implications, and future research directions. Finally, the References section lists all cited works.

2. METHODOLOGY

2.1 Model-Driven Architecture Framework:

A. Overview of MDA:

Model-driven architecture (MDA) is a comprehensive approach to software design, development, and implementation that emphasizes the use of domain models as primary artifacts in the development process. By focusing on creating and exploiting these domain models, MDA provides a systematic framework that helps bridge the gap between the conceptual design and the actual implementation of software systems. One of the fundamental principles of MDA is the separation of the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform. This principle ensures that the core business logic and requirements are captured in a technology-agnostic manner, allowing for greater flexibility and adaptability [2] [3].

At the heart of MDA are two key concepts: the Platform-Independent Model (PIM) and the Platform-Specific Model (PSM). The PIM is an abstract representation of the system, focusing on the essential business logic and functionality without being tied to any technology. This high level of abstraction enables developers to concentrate on the core aspects of the system without being distracted by implementation details. Once the PIM is defined, it can be transformed into one or more PSMs. Each PSM provides a detailed specification of how the system's functionality will be implemented on a specific technology platform, such as a particular programming language, database, or hardware environment [4].

The transformation from PIM to PSM is a crucial aspect of MDA, as it automates the generation of code and other artifacts needed for the implementation of the system. This automation not only improves productivity by reducing the amount of manual coding required but also enhances consistency and reliability by ensuring that the generated artifacts faithfully adhere to the specified models. By maintaining a higher level of abstraction throughout the development process, MDA enables developers to focus on solving business problems and designing robust systems, while the underlying transformations handle the technical details of implementation. This approach leads to more efficient development cycles, better alignment between business requirements and technical solutions, and ultimately, higher-quality software systems [5].

Model-driven architecture (MDA) serves as a pivotal strategy in software design and development, emphasizing the use of models as core artifacts. By implementing MDA, the conversion of UML models into NoSQL database schemas can be automated [6] [7] [8] [9], thereby enhancing consistency and minimizing manual labor. This approach has found applications across numerous fields, including the development of web service platforms [10], web frameworks [6] [7] [8] [9], blockchain and IoT systems [15] [16] [17], artificial intelligence [18], mobile applications [19], and database generation [20] [21].

B. Transformation types in MDA:

In the MDA framework, transformations are categorized into several types based on the direction and nature of the transformation process. The primary types include Model-to-Model (M2M) transformations, which involve converting one model to another within the same level of abstraction, such as transforming a PIM into a PSM by mapping abstract concepts to platform-specific implementations. In the context of this study, the transformation of UML models to MongoDB schemas is an example of an M2M transformation. Model-to-Text (M2T) transformations generate textual artifacts like code, configuration files, or documentation from models, exemplified by generating Java code from a UML model. Text-to-model (T2M) transformations parse textual artifacts and generate models from them, such as creating a UML model from a set of XML configuration files. Additionally, model merging and synchronization processes involve combining multiple models into a single coherent model or ensuring that different models remain consistent with each other over time [2].

C. Key components of MDA:

Key components of Model-Driven Architecture (MDA) include the Platform-Independent Model (PIM), which is an abstract model that defines the system's functionality without concern for the underlying platform, and the Platform-Specific Model (PSM), which includes details about the specific platform and technology used to implement the system. Transformation rules are guidelines and algorithms that map elements from the PIM to corresponding elements in the PSM. Additionally, the Meta-Object Facility (MOF) is a standard for defining metamodels, which are models that describe the structure and semantics of other models.

D. MDA Workflow:

The MDA process depicted in the Figure below demonstrates a structured approach to software development, starting from a business request and progressing through various levels of abstraction to reach platform-specific implementations. This approach ensures that business requirements are accurately captured, abstracted, and systematically transformed into executable models on different platforms. By following this process, developers can maintain consistency, improve productivity, and ensure that the final implementations align closely with the initial business goals and requirements.

Figure 1: Model Driven Architecture Workflow

E. Application of MDA in UML to MongoDB Transformation:

The application of MDA in UML to MongoDB transformation involves several key components. The Platform-Independent Model (PIM) is represented by the UML model, which captures the structure of the system in terms of classes, properties, and associations. The Platform-Specific Model (PSM) is embodied by the MongoDB schema, detailing how data is stored in collections, documents, and fields. Transformation rules defined using QVTo—a model transformation language—map UML elements (classes, properties, associations) to MongoDB elements (collections, documents, fields), ensuring that the semantics and relationships of the UML model are preserved in the MongoDB schema. The transformation process is automated with QVTo scripts, which read the UML model and generate the corresponding MongoDB schema, thereby reducing manual effort, minimizing errors, and ensuring consistency between the PIM and PSM.

F. Benefits of Applying MDA:

The benefits of applying MDA include ensuring consistency by consistently translating the high-level design in the UML model into the MongoDB schema, enhancing efficiency by automating the transformation process and reducing the potential for human error, facilitating flexibility by allowing changes at the model level to be automatically propagated to the database schema, and supporting scalability by enabling the creation of scalable and maintainable database schemas that align with the evolving needs of the application.

2.2 Metamodel Definitions:

A. UML Metamodel:

The UML metamodel defines the structure and semantics of UML models, which are used to capture the design of software systems. Key elements of the UML metamodel include classes, properties, and associations. Classes represent the main building blocks of the system, encapsulating data and behavior. Properties are attributes of classes, defining the characteristics and data stored within each class. Associations define relationships between classes, indicating how instances of one class relate to instances of another.

In this study, the UML metamodel is used to define the structure of the system in terms of its classes, properties, and associations. This metamodel provides the foundation for the Platform-Independent Model (PIM) in our MDA approach, capturing the system's design without concern for the underlying implementation platform.

B. MongoDB Metamodel:

The MongoDB metamodel defines the structure and semantics of MongoDB schemas, which are used to store data in a flexible and scalable manner. Key elements of the MongoDB metamodel include collections, documents, and fields. Collections are analogous to tables in relational databases and are used to group related documents. Documents are JSON-like structures that store data in key-value pairs, representing instances of classes in UML. Fields are individual attributes within documents, corresponding to properties in UML classes.

In our implementation, the MongoDB metamodel is used to define the structure of the database schema that will be generated from the UML model. This metamodel provides the foundation for the Platform-Specific Model (PSM) in our MDA approach, detailing how data is stored and managed in MongoDB.

2.3 QVTo Transformation Process:

A. QVTo:

QVTo (QVT Operational) is a model transformation language specified by the Object Management Group (OMG) for transforming models in a precise and automated manner. QVTo

is part of the QVT (Query/View/Transformation) family of languages and is particularly suited for imperative transformations where complex mappings and control flow are necessary [22]. In the context of this study, QVTo is used to define and execute the transformation rules that map UML models to MongoDB schemas.

B. Mapping rules:

The transformation from UML to MongoDB involves several key mapping rules that ensure the semantic integrity and structural accuracy of the models are preserved. The following are the primary mapping rules used in this study:

Class to Collection: UML classes are transformed into MongoDB collections. Each class in the UML model is represented as a separate collection in MongoDB. For example, the Student class is mapped to a Student collection.

Property to Field: UML properties are transformed into MongoDB fields. Each property within a UML class becomes a field within the corresponding MongoDB document. For instance, the name property of the Student class is mapped to a name field in the Student collection.

Association to Document Relationship: UML associations are mapped to relationships between MongoDB documents. This ensures that the relationships defined in the UML model are accurately represented in the MongoDB schema.

3. IMPLEMENTATION

3.1 Environment Setup:

A. Tools and Software:

The implementation of the UML to MongoDB transformation using QVTo requires a specific set of tools and software to ensure a smooth and efficient process. Below is a list of the tools and software used, along with their versions and configurations:

Eclipse IDE: Eclipse IDE for Java Developers.

QVTo Plugin: Eclipse QVTo (Operational QVT) plugin.

UML Metamodel: UML 2.5.1 metamodel

MongoDB Metamodel: Custom MongoDB metamodel.

Java Development Kit (JDK): JDK 11 or later

3.2 The QVTo Transformation Script:

Developing the QVTo transformation script involves structuring the script to efficiently and accurately transform UML models into MongoDB schemas. The script is organized into several key components: initialization, mapping rules, and validation steps.

The structure of the QVTo transformation script is designed to systematically map elements from the UML model to the corresponding elements in the MongoDB schema. The script begins with the definition of the source and target metamodels, followed by the main transformation function that orchestrates the overall process. Each transformation rule is implemented as a separate mapping function within the script. This modular approach ensures clarity and maintainability of the transformation logic. The script ends with validation steps to ensure that the generated MongoDB schema accurately reflects the original UML model.

The initialization phase of the QVTo script involves setting up the transformation environment and loading the source UML model and target MongoDB metamodel. This step ensures that the necessary resources are available for the transformation process. The script starts by declaring the model types used, specifying the URIs for the UML and MongoDB metamodels. The main function is defined to invoke the transformation logic, and the root objects of the UML model are mapped to the initial transformation rule.

The core of the QVTo script consists of mapping rules that define how UML elements are translated into MongoDB elements. These rules are implemented as QVTo mappings and cover the primary elements in both metamodels. For example, UML classes are mapped to MongoDB collections, properties are mapped to fields within documents, and associations are mapped to relationships between documents. Each mapping rule includes specific logic to extract relevant information from the UML model and create the corresponding MongoDB elements. The use of collection operations ensures that all relevant elements in the UML model are processed and transformed.

Validation is a critical part of the transformation process to ensure the generated MongoDB schema is correct and consistent with the UML model. The QVTo script includes validation steps that check the integrity and accuracy of the transformation results. This involves verifying that all UML classes, properties, and associations have been appropriately mapped to MongoDB collections, fields, and relationships. Additionally, the script performs checks to ensure that the generated schema adheres

to MongoDB's document-oriented data structure requirements. These validation steps help identify and rectify any discrepancies, ensuring the final schema is both structurally and semantically accurate.

By carefully structuring the QVTo transformation script and incorporating comprehensive mapping rules and validation steps, the transformation process from UML models to MongoDB schemas is made efficient, accurate, and reliable. This approach ensures that the high-level design captured in the UML model is faithfully represented in the resulting MongoDB schema, facilitating seamless integration and data management.

3.3 Sample UML Model:

To illustrate the transformation process, a sample UML model is used as a case study. This model represents a simple university system consisting of classes such as Student, Teacher, and Course, each with their respective properties and associations. The Student class includes properties like studentID, name, and email. The Teacher class includes properties such as teacherID, name, and department. Additionally, the Course class includes properties like courseID, title, and credits. This model serves as a representative example to demonstrate the transformation from UML to MongoDB.

Classes and Properties:

In the sample UML model, the Student class encapsulates the attributes of a student, with properties including:

- **`studentID`:** A unique identifier for each student.
- **`name`:** The name of the student.
- **`email`:** The email address of the student.

The **Teacher** class includes properties that describe a teacher:

- **`teacherID`:** A unique identifier for each teacher.
- **`name`:** The name of the teacher.
- **`department`:** The department to which the teacher belongs.

The **Course** class represents a course offered at the university, with properties including:

- **`coursed`:** A unique identifier for each course.
- **`title`:** The title of the course.
- **`credits`:** The number of credits the course is worth, representing the workload and learning value of the course.

Visual Representation:

Here is the UML model representing the Student, Teacher, and Course classes with their respective properties:

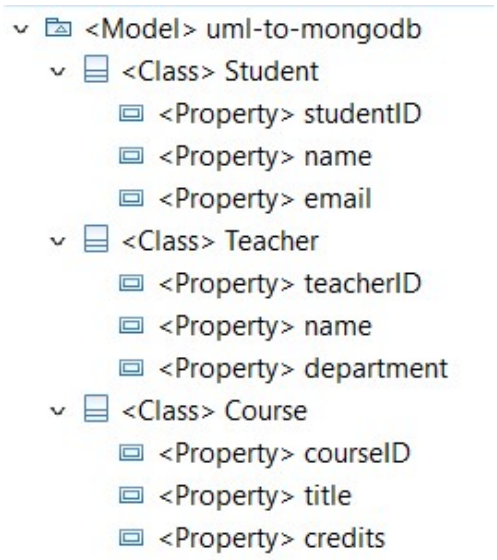

Figure 2: UML Model Representation of a University System with Classes and Properties.

3.4 Executing the Transformation:

Executing the transformation from UML to MongoDB involves several steps, starting with loading the UML model and MongoDB metamodel into the QVTo transformation tool within the Eclipse IDE. The UML model, in XMI format, is loaded as the source model, while the MongoDB metamodel, defining the target schema structure, is loaded as the target model. This setup ensures the QVTo script has access to both metamodels, enabling accurate transformation. As shown in Figure 4, the configuration screen in Eclipse allows specifying the transformation module, trace file generation, and model parameters. The next step is to run the QVTo transformation script from the QVTo perspective in Eclipse. The script reads the UML model, applies the defined transformation rules, and generates the corresponding MongoDB schema. During execution, the script processes each UML model element according to the mapping rules, converting UML classes, properties, and associations into MongoDB collections, documents, and fields. Upon successful execution, the output is an XMI-formatted MongoDB schema, preserving the structure and semantics of the original UML model. This schema includes collections for UML classes, documents for instances of these classes, and fields for the properties of the UML classes. The entire process, from loading models to generating the MongoDB schema, is designed to be seamless and efficient, leveraging QVTo's capabilities to automate the transformation, ensuring consistency, accuracy, and adherence to the specified design, thereby enhancing productivity and reliability in database schema generation.

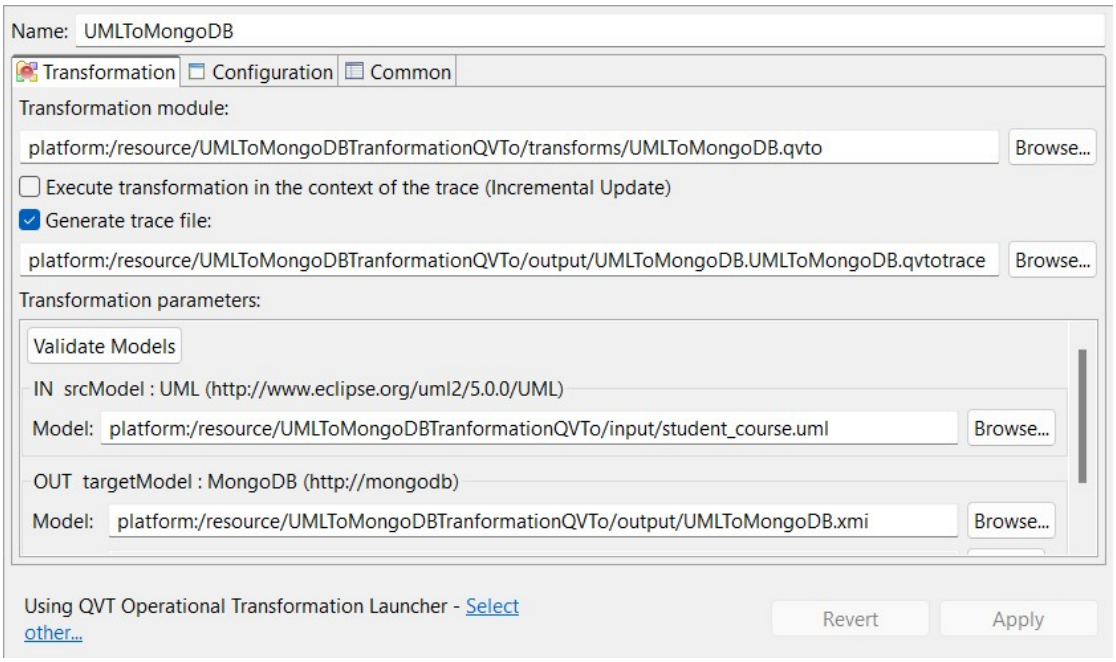

Figure 3: Configuration Screen for Executing QVTo Transformation in Eclipse

3.5 Validation and Testing:

Validation and testing are crucial to ensure that the transformation process accurately converts UML models into MongoDB schemas while preserving the original models' integrity and semantics. The validation process involves checking semantic integrity to ensure the meaning and relationships in the UML model are correctly represented in the MongoDB schema and verifying structural accuracy to ensure elements in the MongoDB schema correspond correctly to elements in the UML model. Key validation criteria include consistency, completeness, and correctness. The testing methodology involves executing test cases designed to validate different aspects of the transformation, including loading test models, running the transformation, comparing results, and checking integrity. Several test cases were used to validate the transformation process, including basic transformations, complex associations, and edge cases. The results analysis showed that most test cases resulted in correct and consistent MongoDB schemas, demonstrating the QVTo script's accuracy. Any discrepancies identified during

testing were analyzed and addressed, ensuring the transformation rules were refined for better accuracy. Overall, the validation and testing demonstrated the QVTo transformation script's reliability and effectiveness in providing a robust, automated solution for converting UML models into MongoDB schemas, maintaining the original UML diagrams' integrity and semantics.

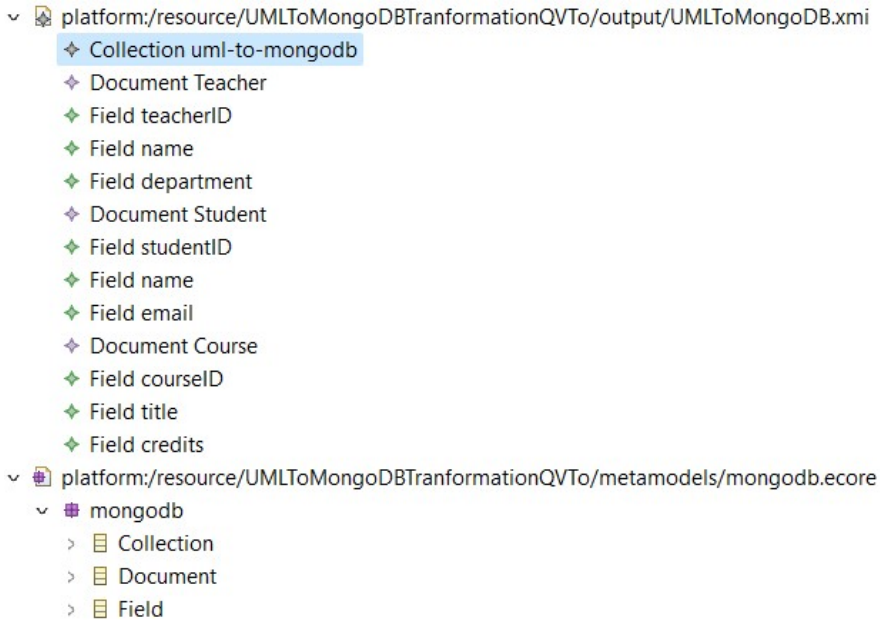

Figure 4: Generated MongoDB Schema from UML Model using QVTo

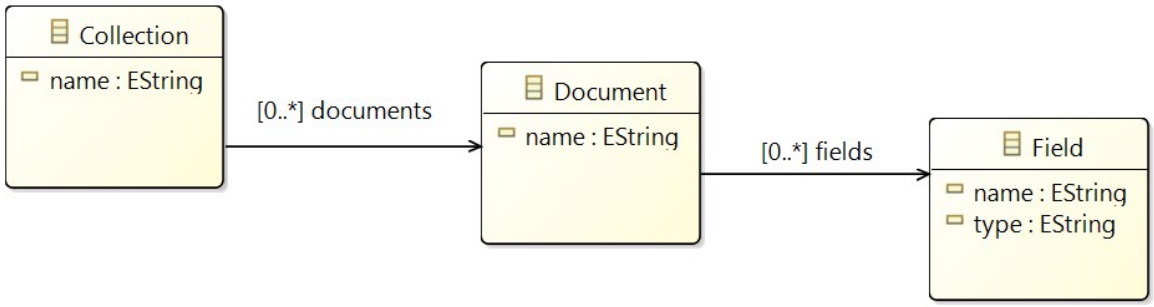

Figure 5: UML Representation of MongoDB Metamodel

UMLToMongoDB.qvto

```
modeltype UML uses uml("http://www.eclipse.org/uml2/3.0.0/UML");
modeltype MongoDB uses mongodb('http://mongodb.mm');

transformation UMLToMongoDB(in srcModel: UML, out targetModel: MongoDB);

main() {
    srcModel.rootObjects()[UML::Model]->map ModelToCollection();
}

mapping UML::Model::ModelToCollection() : mongodb::Collection {
    result.name := self.name;
    result.documents += self.packagedElement[UML::Class]->map ClassToDocument();
}

mapping UML::Class::ClassToDocument() : mongodb::Document {
    result.name := self.name;
    result.fields += self.ownedAttribute[UML::Property]->map PropertyToField();
}

mapping UML::Property::PropertyToField() : mongodb::Field {
    result.name := self.name;
}

```

Figure 6: QVTo Transformation Script for Converting UML Models to MongoDB Schemas

4. CONCLUSION

In this study, we successfully applied Model-Driven Architecture (MDA) to transform UML models into MongoDB schemas using QVTo transformation scripts, addressing the significant challenge of preserving the semantic integrity of UML models in NoSQL databases. Our methodology enhances IT knowledge by introducing novel insights and best practices, filling a gap in existing literature primarily focusing on relational databases. By defining precise metamodels for UML and MongoDB, developing robust QVTo scripts, and validating the transformation process, we automated schema generation, ensuring accuracy and consistency. Compared to previous works, our study offers a comprehensive approach tailored for MongoDB, extending the applicability of MDA to modern, flexible data management systems. This research significantly contributes to model-driven engineering by improving database management efficiency and providing practical guidelines for IT professionals. Future work will explore extending this methodology to other NoSQL databases and integrating advanced features like query optimization and data validation, demonstrating the versatility and scalability of using MDA and QVTo for database schema generation.

REFERENCES:

[1] H. Koç, A. M. Erdoğan, Y. Barjakly, and S. Peker, "UML Diagrams in Software Engineering Research: A Systematic Literature Review," in *The 7th International Management Information Systems Conference*, MDPI, Mar. 2021, p. 13. doi: 10.3390/proceedings2021074013.

[2] D. Mouheb *et al.*, "Model-Driven Architecture and Model Transformations," in *Aspect-Oriented Security Hardening of UML Design Models*, Cham: Springer International Publishing, 2015, pp. 35–45. doi: 10.1007/978-3-319-16106-8_4.

[3] S. J. Mellor, K. Scott, A. Uhl, and D. Weise, "Model-Driven Architecture," in *Advances in Object-Oriented Information Systems*, vol. 2426, J.-M. Bruel and Z. Bellahsene, Eds., in Lecture Notes in Computer Science, vol. 2426. , Berlin, Heidelberg: Springer Berlin

Heidelberg, 2002, pp. 290–297. doi: 10.1007/3-540-46105-1_33.

[4] D. Dayan, R. Kaplinsky, A. Wiesen, and S. Bloch, "AMDA: Matching the Model-Driven-Architecture's Goals Using Extended Automata as a Common Model for Design and Execution," in *IEEE International Conference on Software-Science, Technology & Engineering (SwSTE'07)*, Herzlia, Israel: IEEE, Oct. 2007, pp. 1–13. doi: 10.1109/SwSTE.2007.13.

[5] M. Melouk, Y. Rhazali, and H. Youssef, "An Approach for Transforming CIM to PIM up To PSM in MDA," *Procedia Computer Science*, vol. 170, pp. 869–874, 2020, doi: 10.1016/j.procs.2020.03.122.

[6] F. Abdelhadi, A. Ait Brahim, and G. Zurfluh, "Applying a Model-Driven Approach for UML/OCL Constraints: Application to NoSQL Databases," in *On the Move to Meaningful Internet Systems: OTM 2019 Conferences*, vol. 11877, H. Panetto, C. Debruyne, M. Hepp, D. Lewis, C. A. Ardagna, and R. Meersman, Eds., in Lecture Notes in Computer Science, vol. 11877. , Cham: Springer International Publishing, 2019, pp. 646–660. doi: 10.1007/978-3-030-33246-4_40.

[7] F. Abdelhedi, A. Ait Brahim, F. Atigui, and G. Zurfluh, "MDA-Based Approach for NoSQL Databases Modelling," in *Big Data Analytics and Knowledge Discovery*, vol. 10440, L. Bellatreche and S. Chakravarthy, Eds., in Lecture Notes in Computer Science, vol. 10440. , Cham: Springer International Publishing, 2017, pp. 88–102. doi: 10.1007/978-3-319-64283-3_7.

[8] F. Abdelhedi, A. A. Brahim, and G. Zurfluh, "Towards an automatic approach for implementing UML/OCL models on NoSQL systems".

[9] A. Srai and F. Guerouate, "Towards the Generation of a PSM Model from a PIM Model, Integration of the MDA Approach in NoSQL Databases, the Case of Document-oriented NoSQL Platforms," *IJETT*, vol. 71, no. 5, pp. 146–155, May 2023, doi: 10.14445/22315381/IJETT-V71I5P215.

[10] J. Bezivin, S. Hammoudi, D. Lopes, and F. Jouault, "Applying MDA approach for web service platform," in *Proceedings. Eighth IEEE International Enterprise Distributed Object Computing Conference, 2004. EDOC 2004.*, Monterey, CA, USA: IEEE, 2004, pp. 58–70. doi: 10.1109/EDOC.2004.1342505.

[11] M. Rahmouni, C. Talbi, and S. Ziti, "Model-driven architecture: generating models from Symfony framework," *IJEECS*, vol. 30, no. 3, p. 1659, Jun. 2023, doi: 10.11591/ijeecs.v30.i3.pp1659-1668.

[12] Dr. A. Srai*, Pr. G. Fatima, and Pr. H. D. Lahsini, "Generation of an E-learning Application Through Model Programming," *IJEAT*, vol. 10, no. 2, pp. 195–198, Dec. 2020, doi: 10.35940/ijeat.B2043.1210220.

[13] A. Srai, F. Guerouate, N. Berbiche, and H. D. Lahsini, "Applying MDA approach for Spring MVC Framework," vol. 12, no. 14, 2017.

[14] Ibn Tofail University, Morocco., M. Rahmouni, and S. Mbarki, "Model-Driven Generation of MVC2 Web Applications: From Models to Code," *IJEACS*, vol. 02, no. 07, pp. 217–231, Aug. 2017, doi: 10.24032/ijeacs/0207/04.

[15] M. Abbas, M. Rashid, F. Azam, Y. Rasheed, M. W. Anwar, and M. Humdani, "A Model-Driven Framework for Security Labs using Blockchain Methodology," in *2021 IEEE International Systems Conference (SysCon)*, Vancouver, BC, Canada: IEEE, Apr. 2021, pp. 1–7. doi: 10.1109/SysCon48628.2021.9447125.

[16] M. Jurgelaitis, V. Drungilas, L. Čeponienė, E. Vaičiukynas, R. Butkienė, and J. Čeponis, "Smart Contract Code Generation from Platform Specific Model for Hyperledger Go," in *Trends and Applications in Information Systems and Technologies*, vol. 1368, Á. Rocha, H. Adeli, G. Dzemyda, F. Moreira, and A. M. Ramalho Correia, Eds., in Advances in Intelligent Systems and Computing, vol. 1368. , Cham: Springer International Publishing, 2021, pp. 63–73. doi: 10.1007/978-3-030-72654-6_7.

[17] N. Moadad, I. Damaj, and I. El Kabani, "A Generic MDA-IoT Architecture for Connected Vehicles in Smart Cities," in *2022 IEEE Global Conference on Artificial Intelligence and Internet of Things (GCAIoT)*, Alamein New City, Egypt: IEEE, Dec. 2022, pp. 122–129. doi: 10.1109/GCAIoT57150.2022.10019064.

[18] M. A. Kadampur and S. Al Riyaee, "Skin cancer detection: Applying a deep learning based model driven architecture in the cloud for classifying dermal cell images," *Informatics in Medicine Unlocked*, vol. 18, p. 100282, 2020, doi: 10.1016/j.imu.2019.100282.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[19] H. Benouda, R. Essbai, M. Azizi, and M. Moussaoui, "Modeling and Code Generation of Android Applications Using Acceleo," *IJSEIA*, vol. 10, no. 3, pp. 83–94, Mar. 2016, doi: 10.14257/ijseia.2016.10.3.08.

[20] H. Natek, A. Srai, and F. Guerouate, "Model-Driven Architecture Approach for SQL Generation using Acceleo: A Case Study on MySQL Database," *IJETT*, vol. 71, no. 10, pp. 20–28, Oct. 2023, doi: 10.14445/22315381/IJETT-V71I10P203.

[21] Dr. A. Srai*, Prof. F. Guerouate, and Prof. H. D. Lahsini, "The Integration of the MDA Approach in Document-Oriented NoSQL Databases, the case of Mongo DB," *IJEAT*, vol. 10, no. 3, pp. 115–122, Feb. 2021, doi: 10.35940/ijeat.C2235.0210321.

[22] C. M. Gerpheide, R. R. H. Schiffelers, and A. Serebrenik, "A Bottom-Up Quality Model for QVTo," in *2014 9th International Conference on the Quality of Information and Communications Technology*, Guimaraes, Portugal: IEEE, Sep. 2014, pp. 85–94. doi: 10.1109/QUATIC.2014.18.