PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA MINISTRY
OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

IBN KHALDOUN UNIVERSITY - TIARET

Thesis

Introduced to :

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

To obtain the degree of :

MASTER

Specialty: Computer science Engineering

Presented by:

Merouane Mohamed Elamine

On the subject

Online persistence and analysis of model transformation exampels

Publicly defended on 4 /7/ 2022 in Tiaret before the jury composed of:

Mr BENDAOUD Mebarek	Grade	University	PR	President
Mr SIABDELHADI Ahmed	Grade	University	MAA	Supervisor
Mr ALEM Abdelkader	Grade	University	MAA	Examiner

2022-2023

ملخص

تقدم هذه المذكرة العلمية إطارًا مبتكرًا لإنشاء مجموعات البيانات في بحوث وتطوير علوم الحاسوب. تتناول المذكرة العلمية الإطار المقترح والذي يهدف لمعالجة الإشكاليات المتعلقة بالفعالية وغياب التوحيد والتعاون المحدود في النهج التقليدي. يعتمد الإطار على تحويلات النماذج بواسطة الأمثلة، حيث يتاح للمستخدمين تحديد التحويلات استنادًا إلى احتياجاتهم. يشمل الإطار مستودعًا يتيح مشاركة النماذج وإعادة استخدامها والتحويلات. يتم تيسير التعاون من خلال منصة إلكترونية عبر الإنترنت، حيث يتمكن مصممو النماذج من العمل معًا وتقديم ملاحظات وتحسين جودة مجموعات البيانات. يتم تقييم فعالية الإطار وكفاءته من خلال التجارب ودراسات الحالة وملاحظات المستخدمين. المساهمة الرئيسية للمذكرة هي تقديم إطار شامل يسرع عملية إنشاء مجموعات البيانات، ويعزز جودة المجموعات المنشأة، ويعزز التعاون ضمن مجتمع علوم الحاسوب.

الكلمات الرئيسية: نموذج هندسي ، نموذج ميتا ، تحويل نموذج بالمثال ، إعادة استخدام في النمذجة التعاونية

Abstract

This thesis proposes an innovative framework for dataset creation in computer science research and development. It addresses inefficiencies, lack of standardization, and limited collaboration in traditional approaches. The framework utilizes model transformations by examples, allowing users to define transformations based on their requirements. It includes a repository for sharing and reusing models and transformations. Collaboration is facilitated through an online platform, enabling modelers to work together, provide feedback, and enhance dataset quality. The framework's effectiveness and efficiency are evaluated through experiments, case studies, and user feedback. The main contribution is a comprehensive framework that accelerates dataset creation, enhances dataset quality, and fosters collaboration within the computer science community.

Keywords: **Model driven-engineering, Meta-model, Model transformation by example, reuse in collaborative modeling**

Contents

Abstract 1

Dedication 11

Acknowledgments 12

1 General Introduction 13

1.1 Research Context 13

1.2 Problem Statement 14

1.3 Contributions 15

1.4 Thesis structure 16

2 Related work 17

2.1 Introduction 17

2.2 Basic concepts 17

2.2.1 Model-driven engineering (MDE) 18

2.2.2 Model 18

2.2.3 Meta-model 19

2.2.4 Meta-meta-model 19

2.3 Model transformation 20

2.3.1 Exogenous Transformation 21

2.3.2 Endogenous Transformation 21
2.3.3 Transformation approaches 22
2.4 Why example 23
2.4.1 Example-based approaches 25
2.4.2 Programming by example 25
2.5 Model transformation by example (MTBE) 25
2.5.1 Transformation Rules 27
2.6 Collaboration 33
2.7 Reuse 34
2.8 Conclusion 34

3 Contribution **35**
3.1 Introduction 35
3.2 Examples of modeling language 35
3.2.1 Class Diagram (UML): 35
3.2.2 Relational schema diagram: 36
3.2.3 Systems Modeling Language (SysML): 37
3.2.4 Petri Net : 38
3.2.5 Sequence diagram (UML): 39
3.3 Motivating Example 41
3.4 Coding the source and target model 48
3.5 Meta Modeling via the visual editor 49
3.5.1 Meta-model parts 50
3.5.2 Gathering meta-model parts 54
3.6 Conclusion 55

4 Implementation of our application **57**
4.1 Introduction 57
4.2 Client-server architecture 57

4.3 Technologies used 58

4.3.1 Front-end 58

4.3.2 Back-end 59

4.3.3 DataBase 61

4.4 Application Working 61

4.4.1 MVC design paradigm 61

4.4.2 Illustrate the application 64

4.5 Conclusion 75

5 General conclusion **77**

5.1 Conclusion 77

5.2 Perspectives 78

List of Tables

2.1 Model transformation approaches by example 32

List of Figures

2.1 Modling in MDE 19

2.2 MTBE Process for Exogenous Transformations 21

2.3 MTBE Process for Endogenous Transformations 22

2.4 An example of traces between a class diagram and an entity-relationship model 26

2.5 Concept map of MTBE approaches 30

3.1 Class Diagram (UML) components: concepts, constraints, operation, and relationships . . . 36

3.2 Relational schema diagram components: concepts, constraints, operation, and relationships . 37

3.3 SysML components: concepts, constraints, operation, and relationships 38

3.4 Petri Net components: concepts, constraints, operation, and relationships 39

3.5 Sequence diagram (UML) components: concepts, constraints, operation, and relationships . 40

3.6 Modeling Languages components: concepts, constraints, operation, and relationships 41

3.7 Example of source model (SM) 43

3.8 Scenario of target model (TM1) 45

3.9 Scenario of target model (TM2) 46

3.10 Scenario of target model (TM3) 47

3.11 Example CDSy3 corresponding to a fragment of TMy3 49

3.12 General Meta-model for Modeling Languages 50

3.13 User part 51

3.14 Model Transformation Part 52

3.15 Heuristic part 53
3.16 Source and Target Model Part 54
3.17 Gathering the meta-model parts to build the final form 55

4.1 ReactJs 59
4.2 ExpressJs 60
4.3 ExpressJs 60
4.4 MongoDB 61
4.5 MVC Architecture Pattern 62
4.6 File Structure based on MVC 63
4.7 Register/Login page 64
4.8 Login alert 64
4.9 User Dashboard 65
4.10 Dashboard notification 65
4.11 Initialize model 66
4.12 Describe the model 67
4.13 Create Class 67
4.14 Create Table 68
4.15 Create Relationships between classes 68
4.16 Relationships inform alert 69
4.17 Relationships error alert 69
4.18 Search page 70
4.19 Target Model Representation page 70
4.20 Table Representation 71
4.21 Source Model Representation page 71
4.22 Class Representation 72
4.23 coding a fragment from the model 73
4.24 Collaboration page 73

4.25 Edit page 74
4.26 Class manipulation 74
4.27 Table manipulation 75
4.28 Relationships manipulation 75

Acronyms

MTBE	Model Transformation By Examples
MDE	Model Driven Engineering
MT	Model Transformation
MDD	Model Driven Development
ER	Entity Relationship
OCL	Object Constraints Language
ECL	Embedded Constraint Language
CoSE	Software engineering collaboration
VCS	Version Control System
ILP	Inductive Logic Programming
ARC	Relational Analysis of Concepts
PG	Programmation Genitic

Dedication

I dedicate this thesis to:

All my family members. Dad may Allah bless his soul, Mom may Allah grant her health and long life. My teachers, colleagues, and friends: **Cherif, Youcef, Oussama, Farouk, Khaled, Ali, Habib and AbdelBasset**, and **Karrouch achraf and Toumi Abdelrahman** those who helped me to achieve this work.

Acknowledgments

First of all, praise be to God Almighty and thanked Him for His blessings throughout my research work, which allowed me to complete it successfully. I have tested your guidance day in and day out; you are the one who allows me to finish my studies. I will continue to trust you for my future

I would like to express my gratitude and special thanks to my supervisor **Mr. Siabdelhadi Ahmed** who made this work possible and whose office door was always open whenever I encountered a problem or had any questions about my research or writing, his guidance and advice carried me through all stages of writing my project. It was a great honor and privilege that I work under his supervision. I have been inspired by his dynamism, vision, honesty, and motivation. Thank you sir for your patience with me.

I would like to acknowledge the efforts and contributions of both **Mr. Bendaouad Mebarek** and **Mr. Alem Abdelkader** , as juries in my thesis defense by giving their expertise and constructive criticism have played a pivotal role in shaping the final outcome of my thesis. Their support and guidance have been invaluable to me, and I am truly honored to have had the opportunity to benefit from their knowledge and expertise.

Thank you all

Chapter 1

General Introduction

1.1 Research Context

The field of software engineering is focused on the creation and advancement of intricate, software-based systems. This encompasses the exploration of theories, techniques, and resources for outlining, structuring, examining, and up-keeping software systems. Today's, these software systems have grown considerably in size, complexity, and importance where model transformation is often used in software engineering to transform one model into another as part of the software development process. For example, a high-level design model may be transformed into a low-level implementation model.

Model transformation (MT) refers to the process of converting one model into another model. This can involve changing the format, structure, or content of the model to better suit a particular purpose or system.

The context for model transformation can vary depending on the specific use case like

- **software engineering :** we mentioned above, and this is what we will focus on.
- **Business process modeling :** model transformation serves as a tool to convert a business process model into a workflow model that can be executed.
- **Data integration :** Model transformation is also used in data integration to transform data models from different sources into a common format for analysis or processing.

The transformation of models (MT) has a significant role in facilitating effective communication and collaboration between different parties, increasing the interoperability of systems, and supporting the process of model-driven development(MDD).

1.2 Problem Statement

In the field of computer science, the process of creating datasets for research and development purposes is often time-consuming, resource-intensive, and prone to inconsistencies. Existing approaches for dataset creation lack a streamlined methodology that combines model transformations by examples, model reuse, and collaboration between modelers. This results in a fragmented and inefficient dataset creation process, hindering the progress and effectiveness of computer science research.

Furthermore, the absence of a platform that supports the creation of datasets through model transformations by examples and facilitates collaboration and model reuse limits the sharing and utilization of best practices, hindering the development of high-quality and diverse datasets. This creates a significant challenge for researchers, practitioners, and the broader computer science community, as it restricts the potential impact and advancement of their work.

Therefore, the problem addressed in this thesis is the lack of a comprehensive framework that enables the creation of datasets from model transformations by examples, promotes the reuse of existing models and transformations, and facilitates collaboration among modelers. This problem impedes the efficient and effective creation of datasets and hampers the ability of the computer science community to leverage collective knowledge, leading to suboptimal research outcomes and a lack of standardized approaches for dataset creation.

By addressing this problem, the thesis aims to provide a solution that empowers researchers and practitioners to create datasets more efficiently, encourages the reuse of models and transformations, and fosters collaboration among modelers. The development of such a framework would enhance the quality, diversity, and availability of datasets, enabling the computer science community to accelerate research, validate hypotheses, and drive innovation in various domains.

1.3 Contributions

The main contribution of this thesis is to develop a framework that enables the creation of datasets from model transformations by examples. The framework will empower users to define model transformations based on their individual perspectives or reuse existing transformations created by others. Additionally, the thesis aims to facilitate collaboration among users, allowing them to collectively build and refine datasets through the reuse and collaborative enhancement of each other's models. By achieving these objectives, the thesis seeks to contribute to the computer science community by providing a platform that supports dataset creation, encourages model reuse, and fosters collaboration.

Expected Contributions:

This thesis is expected to make several contributions to the field of model-driven engineering and model transformation:

1. **Dataset Creation through Model Transformations by Examples:** The thesis will explore techniques and methodologies for generating datasets through model transformations by examples. Users will be able to define transformations based on their individual perspectives, specifying the desired output datasets. This approach leverages the intuitive nature of model transformations by examples, allowing users to express their intentions and requirements in a more accessible manner.

2. **Reuse of Existing Model Transformations:** The framework will enable users to reuse model transformations created by others. By providing a repository or a platform for sharing and accessing pre-defined transformations, the thesis will promote the reuse of models and transformations within the community. This reuse aspect enhances efficiency and accelerates the dataset creation process, as users can leverage existing transformations as building blocks for their own datasets.

3. **Collaboration and Community Building:** The thesis will facilitate collaboration between users by providing a platform or online environment where they can collaborate, share insights, and collectively refine their models and datasets. Users will be able to collaborate on transforming models, exchange ideas, provide feedback, and enhance the quality and reliability of the generated datasets. This col-

laborative aspect fosters community building within the computer science domain and encourages the collective advancement of dataset creation techniques.

1.4 Thesis structure

Our thesis is organized into three chapters:

1. **Chapter 1:** General introduction.
2. **Chapter 2:** Model Driven Engineering, model transformation, and model transformation by examples.
3. **Chapter 3:** Presentation of approach.
4. **Chapter 4:** Implementation of application.
5. **Chapter 5:** Finally, the last part presents a synthesis of our contributions, the limits, and the perspectives of our proposal.

Chapter 2

Related work

2.1 Introduction

Model Driven Engineering (MDE), first used mainly in the field of software systems, has led to several significant improvements in the development process of software systems has allowed several significant improvements in the development process of complex systems by focusing on complex systems by focusing on more abstract concerns around the models used than on classical programming. It is therefore a form of generative engineering in which all or part of an application is generated from models. A model is an abstraction, a simplification of a system that is needed not only to understand the modeled system but also to guarantee its correct operation.

In this chapter, we will present the generalities of model transformation.

2.2 Basic concepts

This section aims to give a simplified view of the key concepts and terminology related to MDE, by defining some common concepts such as a model, a meta-model, and a meta-meta-model. What is the relationship between these concepts?

2.2.1 Model-driven engineering (MDE)

Model Driven Engineering (MDE) is an innovative software engineering approach that emphasizes the use of models in software development and maintenance procedures. The primary aim of this methodology is to elevate the level of abstraction in software development, shifting the focus from code-centric to model-centric operations. This is due to the growing complexity of systems, which necessitates a higher level of abstraction [1].

Is a technique that aims to reduce the complexity of developing and managing modern applications through the use of models. Despite being quite a methodology, it is gaining increasing interest from the industry, which considers it a possible solution to the ever-increasing indicators of quality, efficiency and maintainability. It allows you to treat models as data and then use them as primary units of the development process.

According to *Rothenberg* [2]: *"Modeling in its broadest sense is the cost effective use of something in place of something else for some purpose. It allows us to use something that is simpler, safer, or cheaper than reality instead of reality for some purpose. A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality"*.

This definition perfectly describes the principles and benefits of modeling. A model is an abstraction. It is a system simplification sufficient to understand the modeled system. Models simplify the management of systems by presenting requirements and problems in different views. For instance, a class diagram facilitates the comprehension of an application independently from its platform.

This definitions above provides a broad understanding of the principles and practical applications of models. Moving forward,we will focus on the meanings of models in the context of MDE.

2.2.2 Model

Definition 1: *"A model is a description of (part of) a system written in a well-defined language"*[3].

According to the definition, the notion of model explicitly makes reference to the notion of well defined language which defines the language concepts of a model: such a language is defined by a meta-model.

2.2.3 Meta-model

Definition 2: *"A meta-model is a model that defines the language for expressing a model"*[3]. *To handle a model, which is the goal of MDE, the language of the model must be defined. The models written with this language are said to conform to a meta-model. A meta-model is also considered as a model. It conforms to a meta-model: the meta-meta-model.*

2.2.4 Meta-meta-model

Definition 3: *"A meta-meta-model is a model that defines the language for expressing the meta-modeling languages. A meta-meta-model may conform to itself. Thus, each modeling platform has a meta-meta-model, e.g, Ecore [4] is the meta-meta-model of Eclipse, or the meta-object family (MOF) [5] is the meta-model defined by the Object Management Group (OMG), etc."*

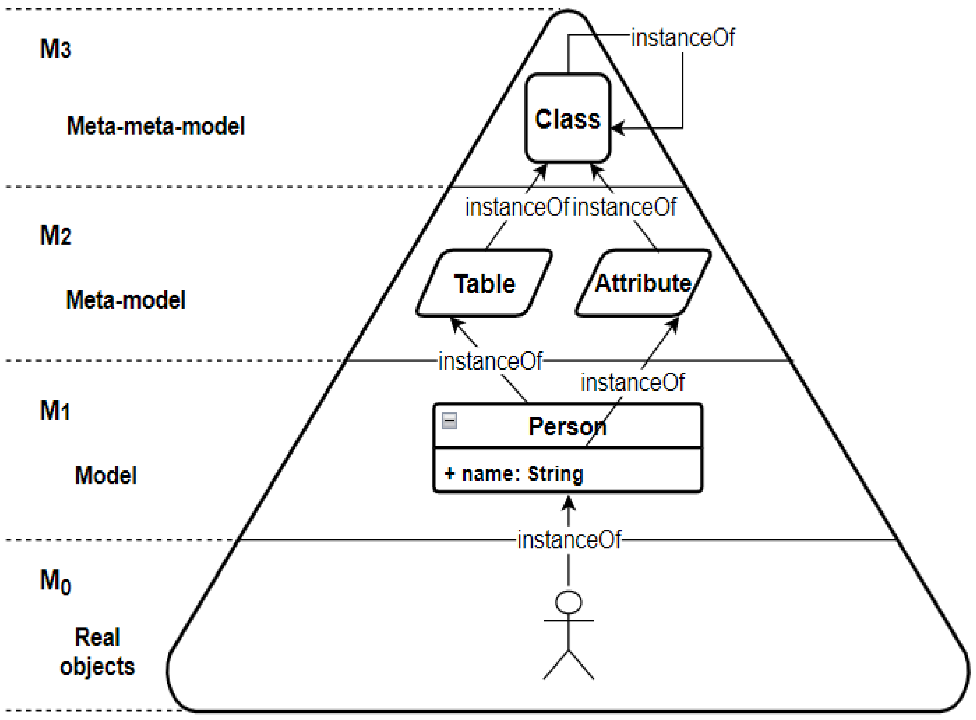


Figure 2.1: Modling in MDE

Figure 2.1 shows an example of modeling. Level M0 consists on the real objects (a person). Level M1 contains the representation which describes the concept of Person with its attribute name. The meta-model of this representation is shown at level M2. It describes the concepts used in M1 (Table, Attribute). These concepts are in turn defined at level M3 which presents the meta-meta-model.

MDE encourages models to have a significant role in the development process of software. Models are written according to meta-models that represent the concepts of the modeling language. For instance, the UML class diagrams define the concepts of class and attribute, the relational schema models define the concepts of table and column, etc.

2.3 Model transformation

In 2003 kleppe et al [3] provide the following definitions of Model transformation :

Definition 4: *"A transformation is the automatic generation of a target model from a source model according to a transformation definition"*.

Definition 5: *"A transformation definition is a set of transformation rules that together describe how a model in a source language can be transformed into a model in a target language"*.

Definition 6: *"A transformation rule is a description of how one or more constructs in a source language can be transformed into one or more constructs in a target language"*.

A model transformation program takes as input a model corresponding to a given source meta-model and produces as output another model corresponding to a target meta-model. A transformation program consisting of a set of rules should itself be considered a model. Therefore, it has a corresponding metamodel, which is an abstract definition of the transformation language used.

we distinguish between two model transformation categories:

2.3.1 Exogenous Transformation

Exogenous transformations are used both to exploit the constructive nature of models in terms of vertical transformations,thereby changing the level of abstraction and building the bases for code generation, and for horizontal transformation of models that are at the same level of abstraction [6]. Horizontal transformations are of specific interest to realize different integration scenarios, e.g., translating a UML class model into an Entity Relationship (ER) model. In vertical and horizontal exogenous transformations, the complete output model has to be built from scratch.[7]

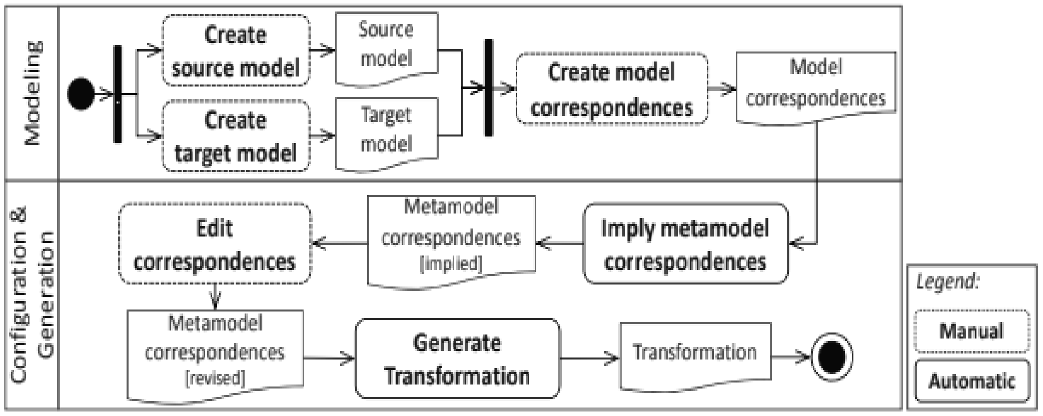


Figure 2.2: MTBE Process for Exogenous Transformations

2.3.2 Endogenous Transformation

In contrast to exogenous transformations, endogenous transformation only rewrite the input model to produce the output model. For this, the first step is the identification of model elements to rewrite, and in the second step these elements are updated, added, and deleted. Endogenous transformations are applied for different tasks such as model refactoring, optimization, evolution, and simulation, to name just a few.[7]

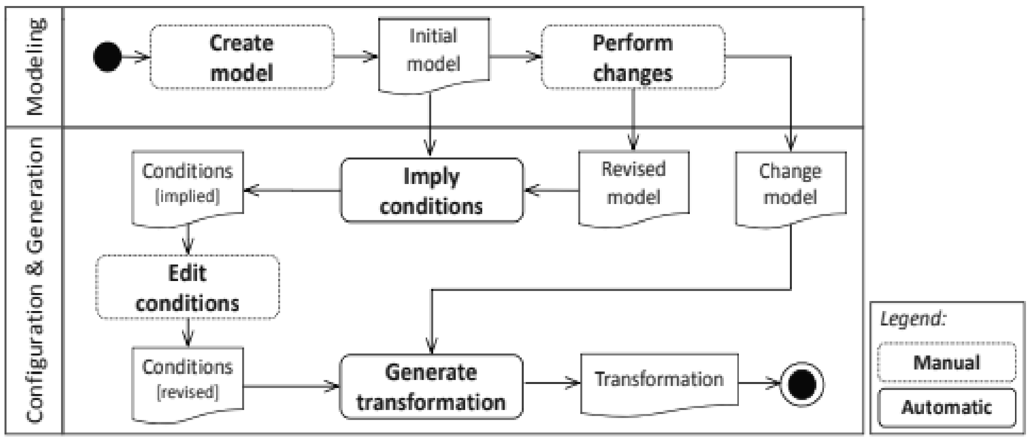


Figure 2.3: MTBE Process for Endogenous Transformations

2.3.3 Transformation approaches

Direct-Manipulation Approaches. These approaches offer an internal model representation plus some API to manipulate it. They are usually implemented as an object-oriented framework, which may also provide some minimal infrastructure to organize the transformations (e.g., abstract class for transformations). However, users have to implement transformation rules and scheduling mostly from scratch using a programming language such as Java. Examples of this approach include Jamda and implementing transformations directly against some MOF-compliant API (e.g., JMI). [8]

Relational Approaches. This category groups declarative approaches where the main concept is mathematical relations.

The basic idea is to state the source and target element type of a relation and specify it using constraints. [8]

Graph-Transformation-Based Approaches. This category of model transformation approaches draws on the theoretical work on graph transformations. In particular, these approaches operate on typed, attributed, labeled graphs [AEH+96], which is a kind of graphs specifically designed to represent UML-like models. Examples of graph-transformation approaches to model transformation include VIATRA, ATOM, GreAT, UMLX, and BOTL.

Similar to relational approaches, graph-transformation approaches are capable of expressing model transformation in a declarative manner. However, the provision of specialized facilities such as graph patterns and graph pattern matching differentiates graph-transformation approaches from the relational ones. [8]

Structure-Driven Approaches. Approaches in this category have two distinct phases: the first phase is concerned with creating the hierarchical structure of the target model, whereas the second phase sets the attributes and references in the target. The overall framework determines the scheduling and application strategy; users are only concerned with providing the transformation rules.

An example of the structure-driven approach is the model-to-model transformation framework provided by OptimalJ. The framework is implemented in Java and provides so-called incremental copiers that users have to subclass to define their own transformation rules. The basic metaphor is the idea of copying model elements from the source to the target, which then can be adapted to achieve the desired transformation effect. The framework uses reflection to provide a declarative interface. A transformation rule is implemented as a method with an input parameter whose type determines the source type of the rule, and the method returns a Java object representing the class of the target model element. Rules are not allowed to have side effects and scheduling is completely determined by the framework. [8]

Operational approach. It is similar to direct manipulation, but provides more specialized support for model transformations. A typical solution for this class is to extend The meta-modeling formalism used and the way to express computations. An example is extending query languages such as OCL with imperative constructs. Examples of systems in this category are Embedded Constraint Language (ECL) [9].

2.4 Why example

The significance of examples in human learning process cannot be overstated. Various learning style theories have been developed that rely on examples to facilitate understanding. To learn more about the most popular learning style theories today

All by-example methods share an emphasis on ease of use and reduced learning curves. the by-example paradigm dates back to 1970, with its roots described in ”Learning Structure Descriptions from Examples” [10]. In fact the main idea of ”Why examples ?”, is to give the software examples of how things are done or what the user expects, and let it do the work automatically.

Our work is based on using and reusing past transformation examples. Various “by-example” approaches have been proposed in the software engineering literature in order to achieve collaboration in the field of engineering.

Examples are essential in understanding modal transformation as they help to illustrate the transformation process and demonstrate the practical applications of the technique.

Here are some of the reasons why examples are important in modal transformation:

- **Illustrating the transformation process:** Examples help to show the transformation process from start to finish, making it easier to understand how the technique works. Seeing the transformation in action can help to clarify any confusing or abstract concepts.
- **Demonstrating the practical applications:** Examples can show how modal transformation can be used in real-world situations. This can help to illustrate the benefits of the technique and its potential uses.
- **Providing a basis for comparison:** Examples can be used to compare different transformation techniques or to compare the results of modal transformation to other methods. This can help to validate the accuracy of the technique and provide a better understanding of its limitations.
- **Encouraging problem-solving skills:** Examples provide an opportunity for learners to practice applying modal transformation to solve problems. This can help to develop problem-solving skills and improve understanding of the technique.

Overall, examples are an important tool in understanding modal transformation, helping to clarify the transformation process, demonstrate practical applications, provide a basis for comparison, and encourage problem-solving skills.

2.4.1 Example-based approaches

Figure 2.4 shows a kind of transformation based on a series of examples. These transformation traces consist of an example of pairs with three identified traces (not all traces are shown). For each trace, the target fragment (bottom) has been determined as corresponding to the source fragment (top). The transformation mappings can be identified by an expert during the design process or retrieved (semi-) automatically using a automatically using an example-based computer processing.[11].

2.4.2 Programming by example

Programming by example is a way to program a software system in its own user interface. its own user interface. The user of the system writes a program by giving an example of what the program should do. The system records the sequence of actions and can execute it again. The programming by example allows the user to create programs without doing conventional programming. conventional programming. A user operates on certain data when creating a program for example, but does not necessarily tell the system why this data was selected. The user can provide this missing information by using data descriptions, which are program operands that specify how to select the data to be data to use when running the program. The system automatically provides reasonable default data descriptions when registering a program; if necessary, the user program; if necessary, the user can later modify the descriptions to reflect his or her reflect their true intentions.[11].

2.5 Model transformation by example (MTBE)

MTBE works to automatically derive a transformation program from a set of examples provided as input. Each sample is a model pair consisting of a source model and a target model. In addition to examples, the majority of MTBE approaches exploit fine traces of transformation. These traces are many-to-many links that associate a group of n source elements with a group of m target elements. In each trace, the target fragment (bottom of the figure) is associated with the source fragment (at the top of the figure). Transformation traces are usually defined by domain experts.

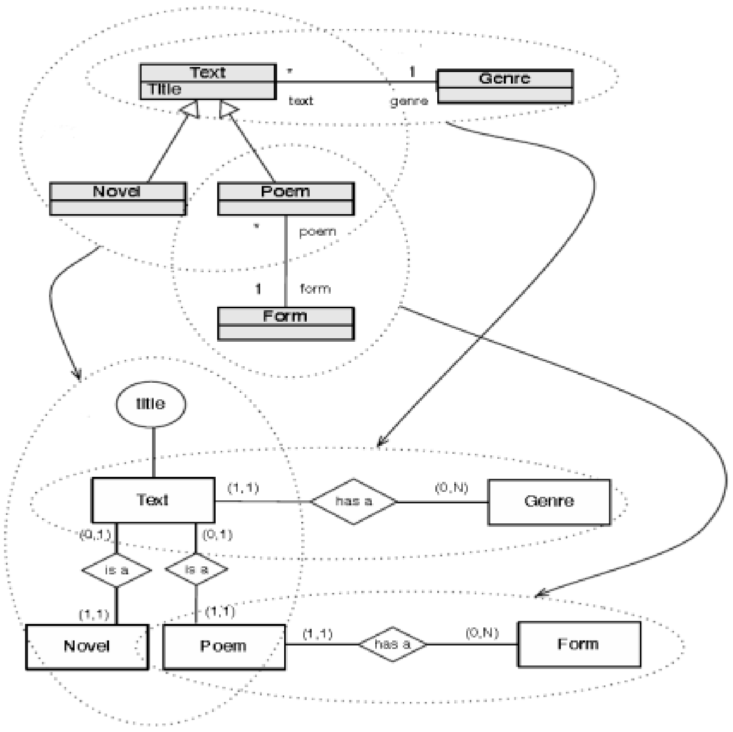


Figure 2.4: An example of traces between a class diagram and an entity-relationship model

Process The main idea of MTBE for exogenous transformations is the semi-automatic generation of transformations from so-called correspondences between source and target model pairs. The underlying process for deriving exogenous model transformations from model pairs is depicted. This process, which is largely the same for all existing approaches, consists of five steps grouped in two phases[7].

Phase 1: Modeling In the first step, the user specifies semantically equivalent model pairs. Each pair consists of a source model and a corresponding target model. The user may decide whether she specifies a single model pair covering all important concepts of the modeling languages, or several model pairs whereby each pair focuses on one particular aspect. In the second step, the user has to align the source model and the target model by defining correspondences between source model elements and corresponding target model elements. For defining these correspondences, a correspondence language has to be available. One important requirement is that the correspondences may be established using the concrete syntax of the modeling languages. Hence, the modeling environment must be capable of visualizing the source and target models as well as the correspondences in one diagram or at least in one dedicated view [7].

Phase 2: Configuration and Generation After finishing the mapping task,a dedicated reasoning algorithm is employed to automatically derive metamodel correspondences from the model correspondences. The automatically derived metamodel correspondences might not always reflect the intended mappings. Thus, the user may revise some metamodel correspondences or add further constraints and computations. Note that this step is not foreseen in all MTBE approaches, because it may be argued that this is contradicting with the general by-example idea of abstracting from the metamodels.

Nevertheless, it seems to be more user-friendly to allow the modification of the metamodel correspondences in contrast to modifying the generated model transformation code at the end of the generation process. Finally, a code generator takes the metamodel correspondences as input and generates executable model transformation code [7].

2.5.1 Transformation Rules

Varró [12] originally put up an MTBE method in 2006. In his work, he starts with a series of archetypal transformation cases that have linked models and then develops transformation rules from them.

The user provides the examples. The analysis of the mappings between the source and target example models, as well as their corresponding meta-models, is the first step in this semi-automated and iterative procedure. An ad hoc algorithm is eventually used to construct the transformation rules.

Varró's method can only provide 1-to-1 transformation rules, which is a significant drawback given that most transformation issues call for n-to-m rules. The rules produced by this method can evaluate if certain components in the source models are present or not. Such a strong sense of expression, while sufficient for many transformation issues, falls short for many widely used issues, such as the conversion of a UML Class diagram into a Relational model. Although source and destination model examples from earlier manual transformations might be used, it is challenging to ensure that the embedded model structures have fine-grained semantic similarity. Lastly, Varró chose not to share the validation data, thus we are unaware of how the method would operate in a real-world situation.

A method that generates 1-to-1 transformation rules based on transformation examples and their traces was put out by Wimmer et al [13]. in 2007 in [7]. With the exception of Wimmer's ability to build executable ATL [13] rules, the derivation procedure is identical to that suggested by Varró in [12]. As the majority of the current transformation issues cannot be solved, providing only 1-to-1 transformation rules is a significant restriction on the application of the technique. Again, the absence of a validation stage prevents evaluation of how the strategy operates in real-world scenarios.

In 2008, Strommer et al [14]. expanded on Wimmer's earlier strategy by allowing 2-to-1 rules in a pattern-matching-based derivation process. In terms of implementation, the authors created a model transformation Eclipse plug-in prototype. This contribution, like the earlier ones, has not been tested in actual conditions, therefore it is still unknown how well it will work in actual circumstances.

Kessentini et al [15] use analogies to perform a transformation. Unlike the contributions cited above, this approach does not produce transformation rules, but rather derives the target model directly from the source model by treating model transformation as an optimization problem. The problem as posed is addressed using particle swarm optimization

The contribution is then further automated by Balogh et al [16] using inductive logic programming (ILP). This new approach, although still iterative and incremental, allows for deriving more complex (n - m) transformation rules. more complex transformation rules (of type n - m).

Another work that fits in the context of model transformations by example is that of Garcia-Magarino et al [17] who propose an algorithm to generate n - m transformation rules, presumably in several transformation languages, from a set of interconnected source and target model pairs (decorated by traces). Like [12], the approach uses graphs. In order to overcome the fact that some transformation languages do not allow writing many-to-many rules, the rules are first derived in a generic transformation language before being transformed into the desired transformation language.

Another MTBE contribution that did not initially result in transformation rules is that of Dolques et al. [18]. This work is based on a relational analysis of concepts (ARC), a variation of the formal analysis of ideas, to categorize the sources, targets, and entry-level correspondences. less clients Identifications are arranged in partially ordered trill and analyzed to select the most relevant ones.

In 2012, Saada et al. [19], extend the work of Dolques et al. [18] by proposing a two-step rule derivation process. In the first step, Dolques' approach is used to learn transformation patterns from transformation examples and transformation traces. In the second step, the learned patterns are analyzed and those considered as pertinent are selected. Selected patterns are, then, translated into transformation rules in JESS language. As in the produced rules are n-to-m. The approach is finally tested in a ten-fold experimental setting where precision and recall are calculated.

The most recent works in the framework of MTBE are those of Faunes et al.[20], in which programmable genetic evolution (PG) is used to make a population of transformations evolve over several generations to produce the desired transformation. The derivation procedure only accepts pairs of examples of source and target models (without any signs of transformation) and outputs rules that can be executed of type n m. The method is improved by Baki et al[21] . where the genetic program attempts to learn both the transformation rules and the execution control that must be applied to them in order to construct an appropriate transformation program simultaneously. This second version also allows for the derivation of rules.

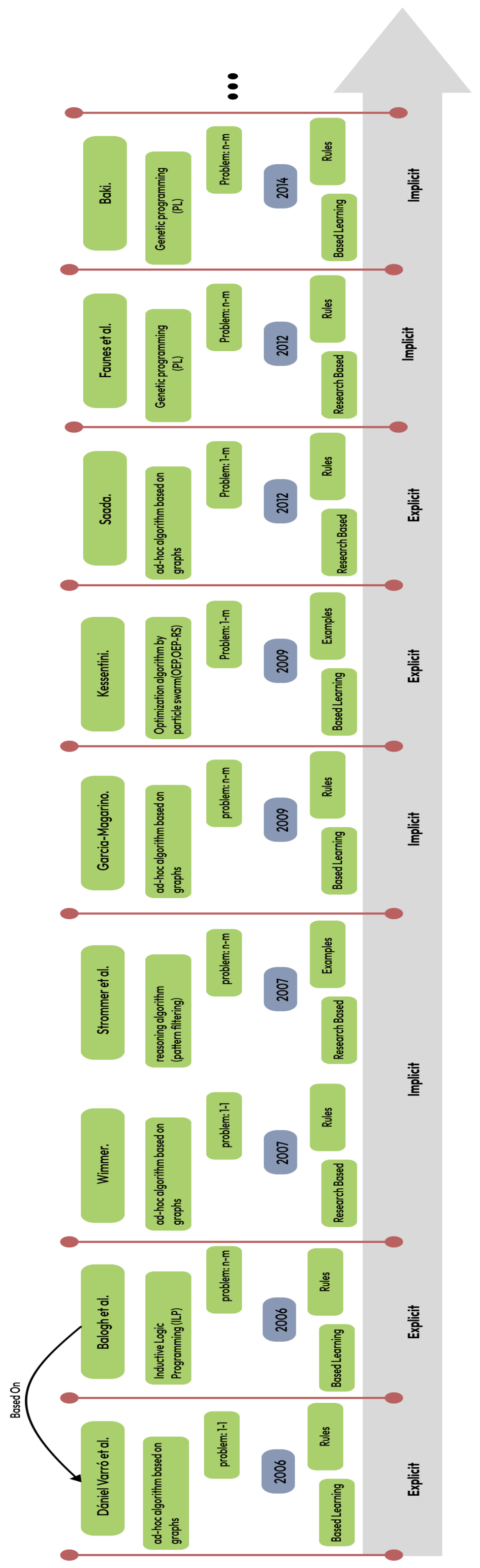


Figure 2.5: Concept map of MTBE approaches

Summary

Table 2.1 summarizes the main aspects of the related work on general model transformation.

The table aims all the approaches that derive transformation rules use transformation example pairs and, with the exception of one work, all of them use transformation traces. They use several derivation processes e.g. pattern matching and heuristic searches. Less than a half of these approaches derive full operational rules, the others do not. With the exception of two works, none of them derives n-to-m rules. None of them, neither, are able to derive the target model by performing advanced operations like concatenating two source model identifiers. They do not implement any kind of rule execution control different than the default (i.e. every rule triggers when matching the source pattern) and with the exception of two contributions, none of them presents a comprehensive validation step. The approaches that derive mappings and other kind of equivalences, also use transformation examples with traces. One of them does not use metamodels since it derives a transformation equivalence at a model level. They also use several derivation processes. Both of them derive n-to-m matchings between source and target models and none of them allows advanced operations or implements execution control. One of them performs a comprehensive validation step.

Approach	Algorithm	Input	Output	N–M rules	Control	Context	Value Con-straints	Complex derivations
Approaches by example								
Varró [12]	Ad-hoc	Examples and Traces	Rules	1-1	No	No	No	No
Wimmer et al.[13]	Ad-hoc	Examples and Traces	Rules	1-1	No	No	No	No
Strommer et al.[14]	Filtering by pattern	Examples and Traces	Rules	N-M	No	No	No	Mentioned only
Kassentini et al.[15]	OEP/OEP-RS	Examples and Traces	MC	-	No	No	No	No
Balogh et al.[16]	PLI	Examples and Traces	Rules	N-M	No	No	No	No
Garcia et al.[17]	Ad-hoc al-gorithm	Examples and Traces	Rules	N-M	No	No	No	No
Kassentini et al.[15]	OEP-RS	Examples	Rules	1-M	No	No	No	No
Deloques et al.[18]	ARC	Examples and Traces	Patterns	-	No	No	No	No
Saada et al.[19]	ARC	Examples and Traces	Rules	1-M	No	No	No	No
Faunes et al.[20]	PG	Examples	Rules	N-M	No	No	No	No
Baki et al.[21]	PG	Examples	Rules	N-M	Yes	Yes	No	No

Table 2.1: Model transformation approaches by example

2.6 Collaboration

Collaboration is a critical component in the field of software engineering. This is due to the following reasons:

- **Improved quality:** Collaboration can help to improve the overall quality of software. When developers work together, they can share their knowledge and expertise, identify and fix bugs more quickly, and ensure that the software meets the highest standards of quality.
- **Better problem-solving:** Collaboration can also help to improve problem-solving. When developers work together, they can bring different perspectives and ideas to the table, which can help to identify and solve problems more effectively.
- **Innovation:** can also foster innovation in the software engineering field. When developers work together, they can come up with new and creative ideas that can help to push the boundaries of what is possible in software development.
- **Increased efficiency:** can also help to increase efficiency in the software development process. When developers work together, they can divide tasks and work on different parts of the project simultaneously, which can help to save time and speed up the development process.

collabirative software engineering (CoSE) deals with methods, processes and tools for enhancing collaboration, communication, and co-ordination (3C) among team members [22].

Software engineering collaboration is model-based, software project management creates organizational structures that encourage collaboration, and global software engineering challenges add complexity to collaboration. Very important to this study is the first important finding, which emphasizes that model-based Nature of CoSE. Massive Collaboration in Software Engineering Refers to software-related artifacts (such as UML diagrams, source code, and bug reports), most of the collaboration in software engineering is over a set of formal or semi-formal artifacts co-workers collaborate upon.

2.7 Reuse

Reusing libraries and components are common practice. most programs are Created by merging several existing libraries, ranging from standard A library of custom modules built in-house. strong cooperation Github and other mainstream platforms and special language features, Polymorphism, for example, facilitates code reuse in these environments. Most software developers rely on VCS, libraries or repositories cooperate.

Reuse dimensions. Alam et al(2018) [23]. Study Shows and Summarize the importance of reusing in order to archive collaboration in the following points :

- **Communication :**category describes support for communication within the environment.
- **Versioning :**category describes support for managing versions of models.
- **Repository :**category describes support for storage, retrieval, ortracking of models.
- **Search :**category describes support for searching models. The tags can be any combination of the following.
- **Granularity :** category describes the granularity of model that may be reused in the environment (this category considers only environment support and not language level support).
- **Relationships :**category describes support for analyzing relationships of models to support, for example, understanding of existing systems or impact of proposed changes.

2.8 Conclusion

In this chapter we have cited the different contributions in the context of learning to transform models by example. And for the summary of these approaches, we have classified in a comparative table that describes the most comparative table that describes the most used approaches in MTBE and summarized in this chapter.

Chapter 3

Contribution

3.1 Introduction

Creating a meta-model for all modeling languages is a complex task that requires a deep understanding of various modeling paradigms, modeling languages, and modeling tools.

3.2 Examples of modeling language

We will do a comparative analysis of the most five common modeling languages listed below in order to extract the common concepts in each that guide us to well define the general meta-model that's fit all modeling languages.

3.2.1 Class Diagram (UML):

UML is a general-purpose modeling language used primarily in software engineering. It is used to visualize, specify, construct, and document the artifacts of a software system. UML is a standardized language and is widely used in industry.[24]

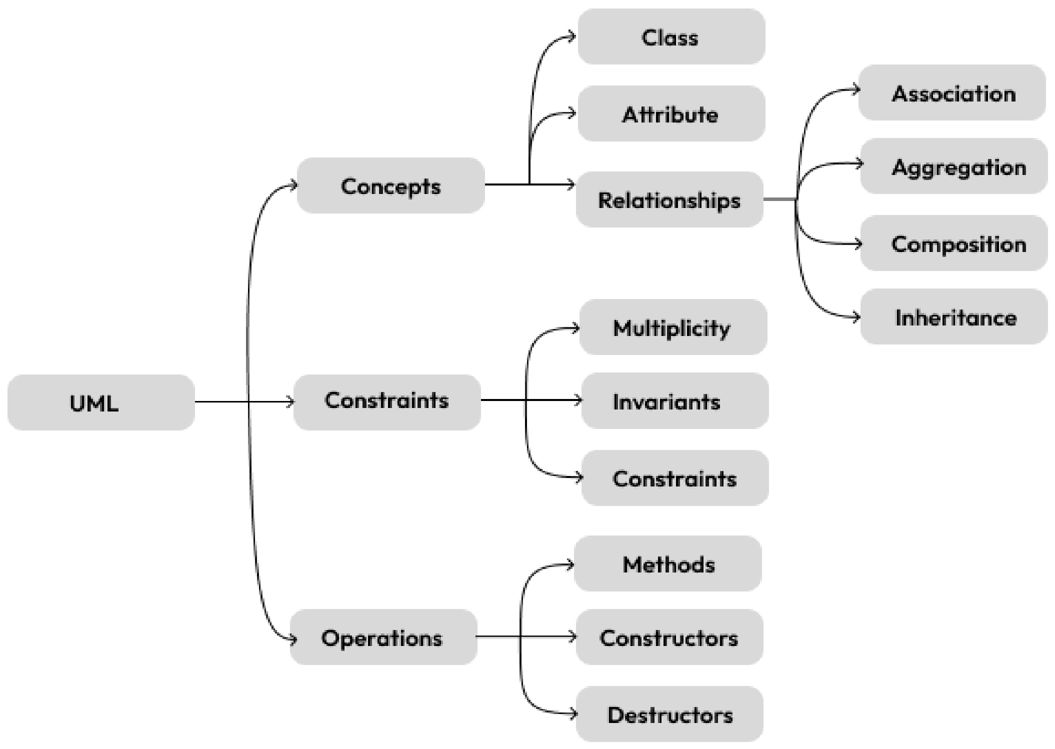


Figure 3.1: Class Diagram (UML) components: concepts, constraints, operation, and relationships

3.2.2 Relational schema diagram:

is used to illustrate the structure of a relational database. It represents the tables (also called relations) in the database and shows the relationships between them through the use of lines connecting the tables. The primary focus of a relational diagram is to demonstrate the tables and their attributes (columns), as well as the primary and foreign key relationships between tables. It provides an overview of the database schema, indicating the tables and their associations, but it does not provide detailed information about the attributes or data types within each table.[25]

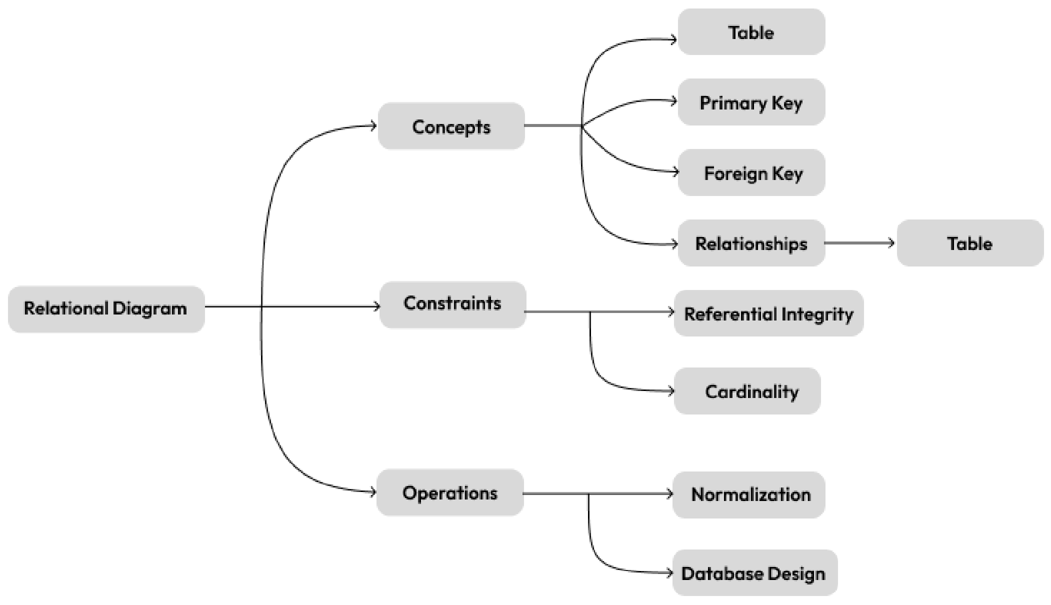


Figure 3.2: Relational schema diagram components: concepts, constraints, operation, and relationships

3.2.3 Systems Modeling Language (SysML):

SysML is a modeling language used to model complex systems. It is used to specify, design, and analyze systems that may include hardware, software, and other components. SysML is a standard language and is widely used in systems engineering.[26]

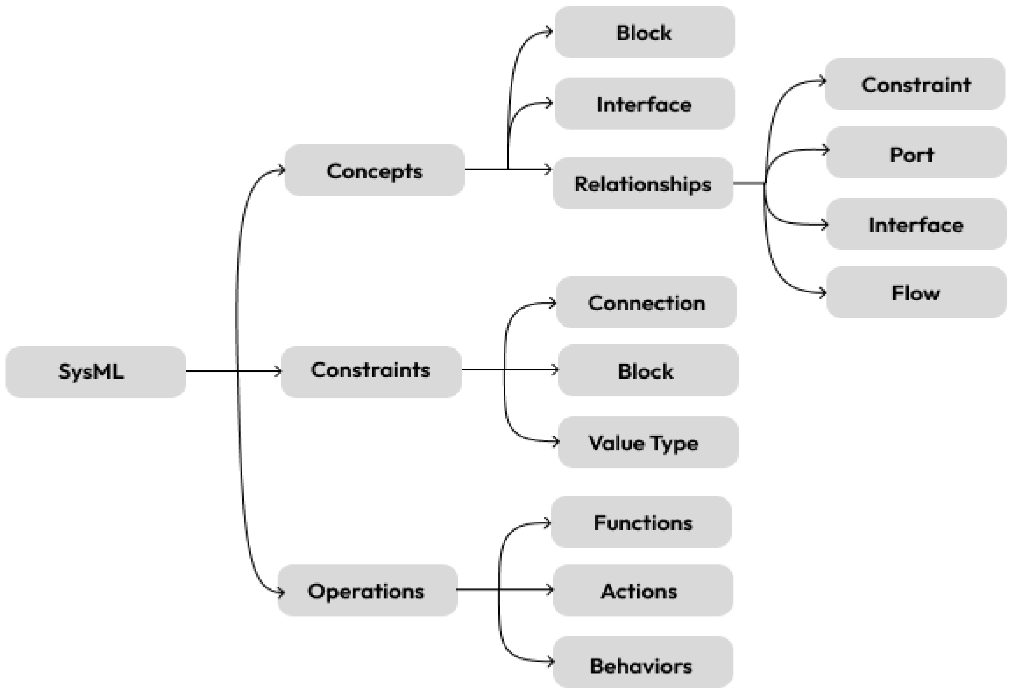


Figure 3.3: SysML components: concepts, constraints, operation, and relationships

3.2.4 Petri Net :

Petri net is a graphical modeling tool used to represent and analyze the behavior of concurrent systems. It consists of places, transitions, and arcs that connect them. Places represent states or conditions, while transitions represent events or actions. The arcs indicate the flow of tokens, which represent resources or events, between places and transitions. Petri nets are used to model the interactions and dependencies between different components in a system and to study their dynamic behavior. They can help identify potential deadlocks, bottlenecks, or other issues that may arise in a system. Petri nets provide a visual and intuitive way to understand how different elements of a system interact and can be used to simulate, analyze, and optimize system processes. They are widely used in various fields, including software engineering, manufacturing, and workflow management, to model and understand complex systems.[27]

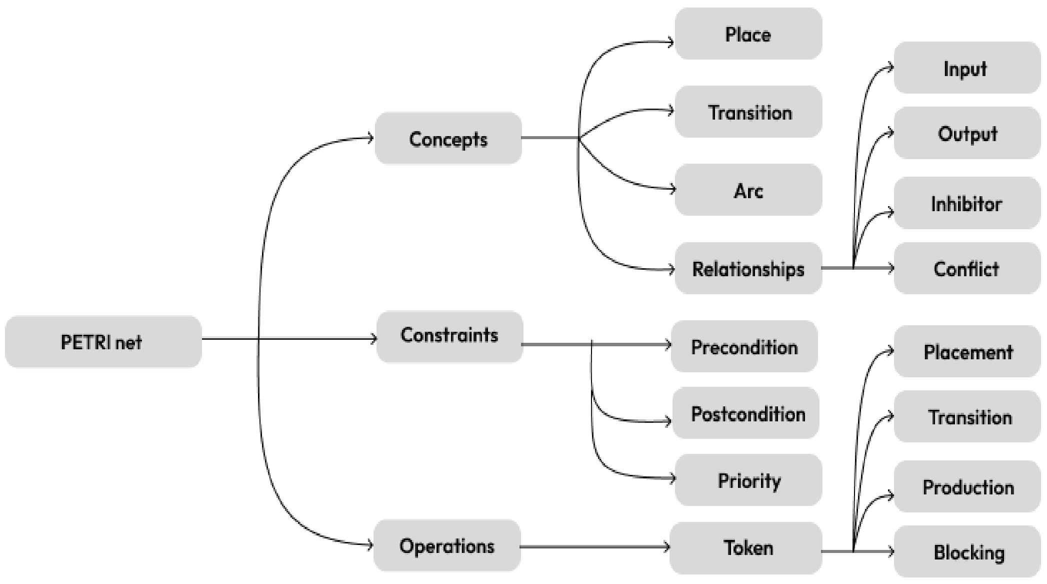


Figure 3.4: Petri Net components: concepts, constraints, operation, and relationships

3.2.5 Sequence diagram (UML):

A sequence diagram is a type of diagram used in Unified Modeling Language (UML) to show the interactions between objects or participants in a system. It focuses on the order of messages exchanged between these objects, representing the flow of control and data during the execution of a particular scenario or use case. In a sequence diagram, objects or participants are represented as vertical lines called lifelines, and messages are depicted as arrows between these lifelines. The diagram shows the timeline of events, with activation bars indicating when objects are actively processing messages. By using sequence diagrams, we can visualize how different objects collaborate, communicate, and respond to each other to achieve specific functionalities in the system. It provides a clear and concise way to understand the dynamic behavior of a system and is useful for analyzing, designing, and documenting software systems.[24]

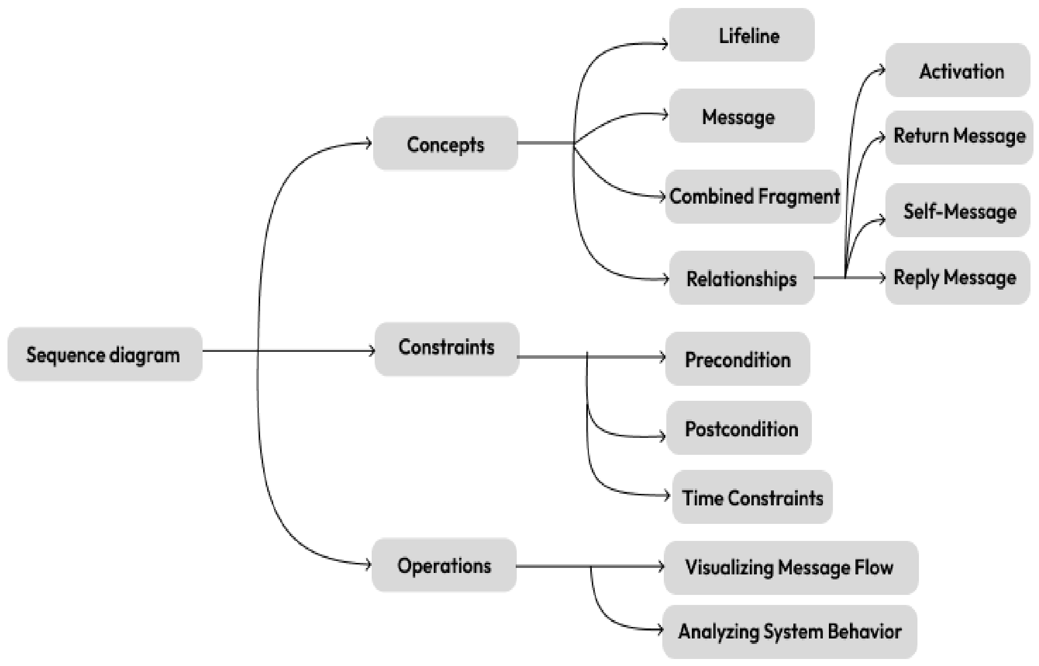


Figure 3.5: Sequence diagram (UML) components: concepts, constraints, operation, and relationships

Remark: If we raise the level of conception, we must delete from each image the variable elements, and it remains just the constant elements from each image, and we have every image that represents a programming language, and thus we can deduce a pattern for all modeling languages consisting only of the static elements of each language.

Figure 3.6 Explain the method used in the Remark section above

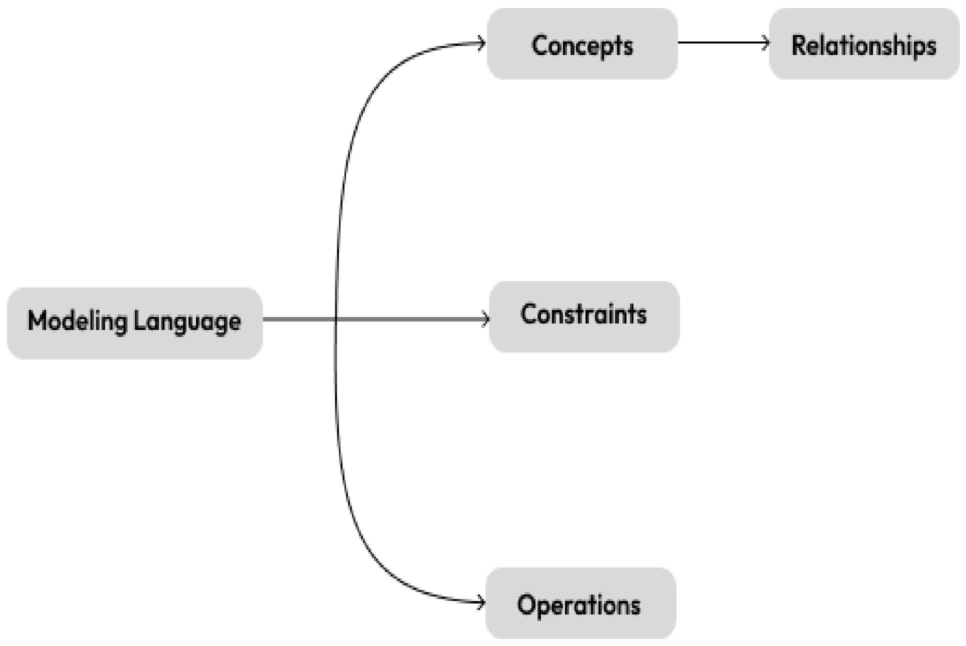


Figure 3.6: Modeling Languages components: concepts, constraints, operation, and relationships

3.3 Motivating Example

We present here a motivating example for our research work to highlight the contributions of our approach for the development of quality-driven model transformations, and especially to clarify different designers' viewpoints in model transformation (MT) by justifying the logic behind each proposal. Figure 3.7 presents an example to highlight different designers' MT scenarios to transform a UML class diagram SMx (i.e. representing a sales model: Customer, Supplier, Part, Date) into a relational model to be persisted in a Relational Database System and satisfying some QoS attributes. The selection of a TM is based on the goal of model transformation that represents user's requirements.

It is made regarding the satisfaction of one or more QoS performance attributes such as the query execution time, the consumed energy, and the maintenance or the storage space. The considered QoS performance attributes are represented by several heuristics to be satisfied by the candidate TMs, each heuristic is associated to one or more QoS performance attributes in a way it can contribute to its achievement.

Our motivating example consists of transforming a UML SM (i.e. SMx, the fragment of the SM shown in gray color in Figure 3.7) into a relational model regarding some performance metrics (e.g. response time, storage size, energy consumption). Let us consider now TMy1 , TMy2, and TMy3 three model transformation scenarios of SMx based on different Qos considerations.

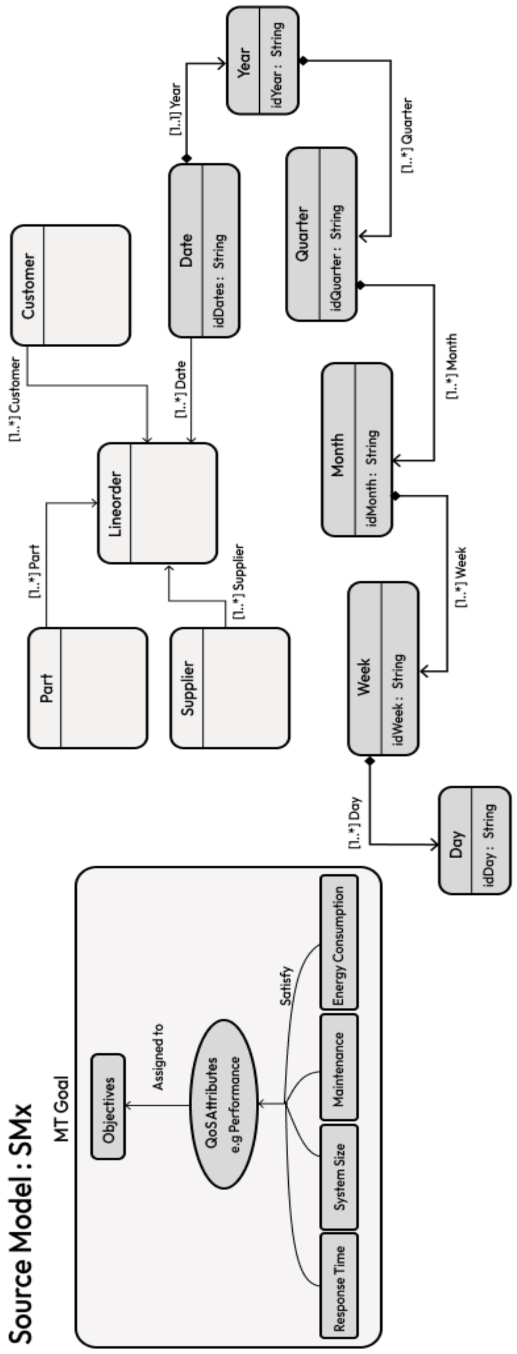


Figure 3.7: Example of source model (SM)

Scenario 1:

In the first scenario, the modeler follows a clear approach of mapping each composite class in the system to its corresponding relation. This mapping is specifically designed for analytical reports, where the user, typically a decision maker, needs to conduct a thorough analysis of sales data at various levels such as Quarter, Month, Week, and Day.

The primary goal in this scenario is to ensure certain quality of service (QoS) attributes are met. These QoS attributes include reducing the storage cost, minimizing maintenance overhead, and optimizing the response time performance for highly selective queries. To achieve these objectives, a Multidimensional Table (MT) designer employs a specific heuristic.

The heuristic employed is to transform each class within the system into its corresponding relation, which must adhere to the third normal form (3NF). By structuring the data in this way, the designer aims to reduce data redundancy, eliminate anomalies, and enhance the efficiency of query execution.

Overall, this approach allows the designer to effectively address the QoS performance attributes while ensuring the data is properly organized and normalized according to the 3NF principles. By adopting this strategy, the system can provide the necessary data analysis capabilities required for in-depth sales analysis, while optimizing storage, maintenance, and query performance.

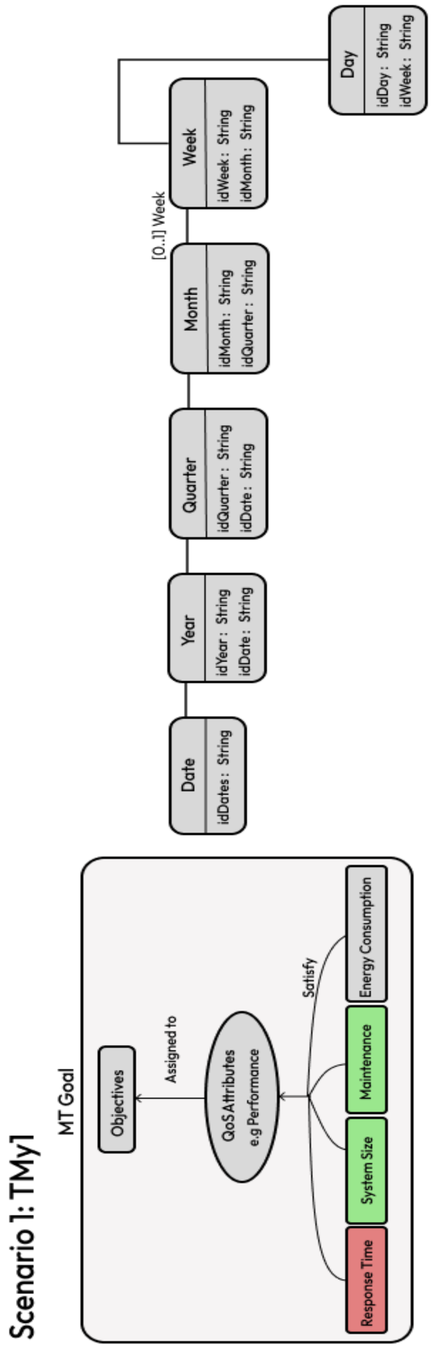


Figure 3.8: Scenario of target model (TM1)

Scenario 2:

In the second scenario, the MT designer merged all SMx's classes in a unique relation (see Figure 3.7). This scenario promotes only one Qos performance attribute, namely, the response time but does not satisfy the two others, the storage space constraint and the maintenance overhead. In this case, the considered heuristics are reducing the number of joins required for insert, update, and delete statements.

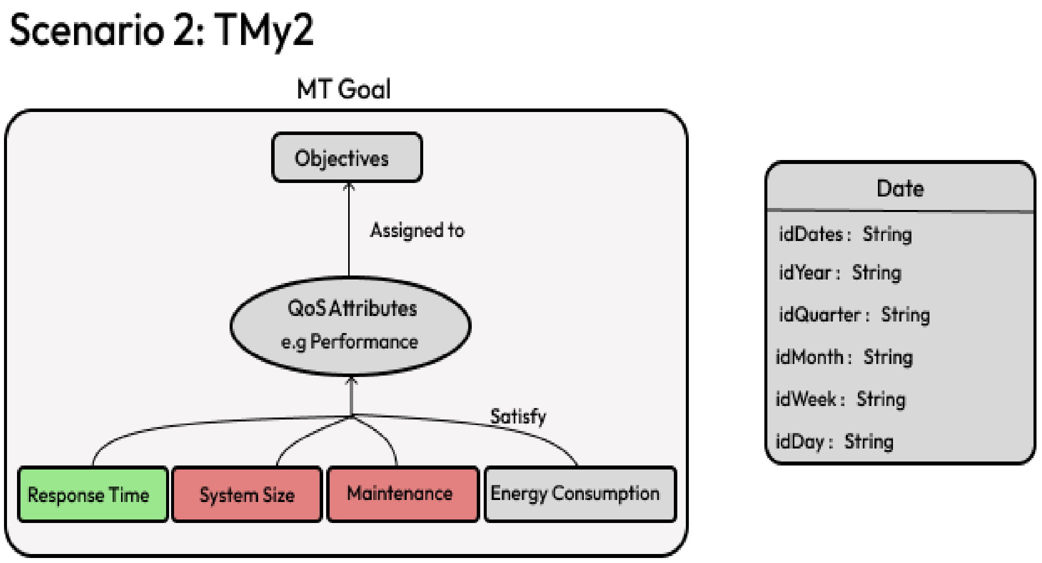


Figure 3.9: Scenario of target model (TM2)

Scenario 3:

In the third scenario, the designer proposes a compromise to satisfy many QoS performance attributes (response time, storage space, and maintenance). In this scenario, the designer considers the following heuristic: Merging/Splitting the SMx's classes regarding a defined threshold number of relations.

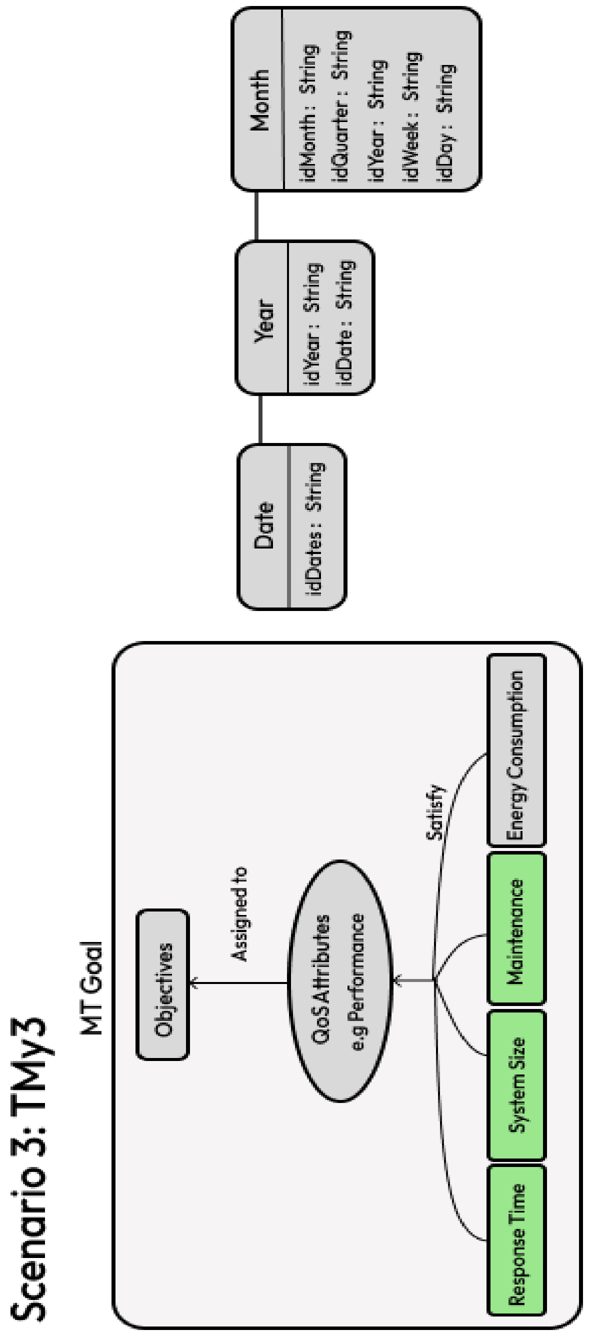


Figure 3.10: Scenario of target model (TM3)

From this motivating example we can see that the designers' transformation viewpoints can be contradictory or/and complementary, this can be justified by the process of model transformation that is driven by heuristics of MT experts in order to satisfy some QoS attributes. Also, the integration of solutions with a refinement process may provide an optimal solution based on domain assumptions or heuristics considered by the MT designers for developing quality-driven model transformations.

The final solution arises from the confrontation between holding different points of view about MT decision. It can be categorical, but also a well-mixed balance.

3.4 Coding the source and target model

Our MTBE approach first requires that MTs be encoded in the form of a feature vector capable of manipulation. Formally, we represent the encoding of a given MTi as a Coding Sequence structure: CDSi = $< E_i, S_i^E, R_i^E, l_i^{SE} >$ where we identify three subsets of objects E = Class, Attribute, Association corresponding to the modifiable objects of TM (i.e. objects that can be added, deleted, and replaced). These objects are divided into properties/sub-properties S_i^E (e.g. association constrained multiplicities). The relationship R_i^E represents the relationship (i.e., association, aggregation, composition, dependency, and inheritance) between the elements of E,(l^{RE}: $E_i \longrightarrow E_j, E_i, E_j \in ExE$), for example. class B is a subclass of class A. The label function l^{Si^E} assigns a unique string label to each sub-property S_i^E For example, we can encode "the type of an attribute" by the value "1" if it is a Key attribute and the value "0" otherwise. Moreover the relation R_{i^E} must retain the model transformation traces that produce TMi of the given SMi.

Finally, our encoding idea allows us to generate the CDS of each TMi solution according to its TMM By therefore, the method used to code a TM solution must be tailored to each DSL. Indeed, our framework serves as recommendations to the developer who must manually refine the coding to be be specific to the DSL in question. Figure 3.11 shows an example of a CDS (CDSy3) corresponding to a fragment of TMy3 from the motivating example.

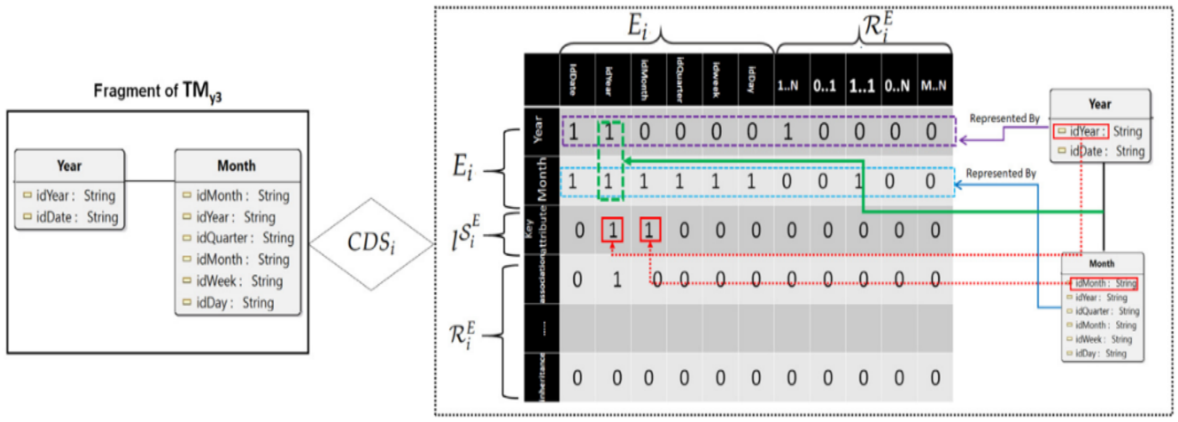


Figure 3.11: Example CDSy3 corresponding to a fragment of TMy3

3.5 Meta Modeling via the visual editor

This meta-model we are presenting is derived from a generalization of the common characteristics found in the constants part of various modeling languages that we have analyzed in the previous section of this chapter. By examining multiple modeling languages, we have identified shared elements and patterns that are essential to the constants part of these languages.

Figure 3.6 provides a visual representation of the meta-model, showcasing its key components and their relationships. The purpose of this meta-model is to provide a unified and comprehensive framework that captures the essential elements and their relationships in the constants part of modeling languages.

By generalizing these common characteristics, we aim to create a higher-level abstraction that can be applied across different modeling languages. This allows us to analyze and compare the constants part of various modeling languages in a more systematic and structured manner.

The meta-model serves as a valuable tool for understanding the underlying structure and concepts in the constants part of modeling languages. It provides a foundation for further analysis, evaluation, and improvement of modeling languages. Additionally, it enables the identification of similarities and differences among modeling languages, which can inform the development of new modeling languages or the enhancement of existing ones.

By leveraging this meta-model, researchers, designers, and practitioners in the field of modeling can gain a deeper understanding of the constants part of modeling languages and make informed decisions regarding language design, usage, and interoperability.

In summary, the meta-model presented in Figure 3.6 represents a generalized view of the constants part of various modeling languages, providing a comprehensive framework for analysis and comparison. It forms a solid basis for further exploration and advancement in the field of modeling languages.

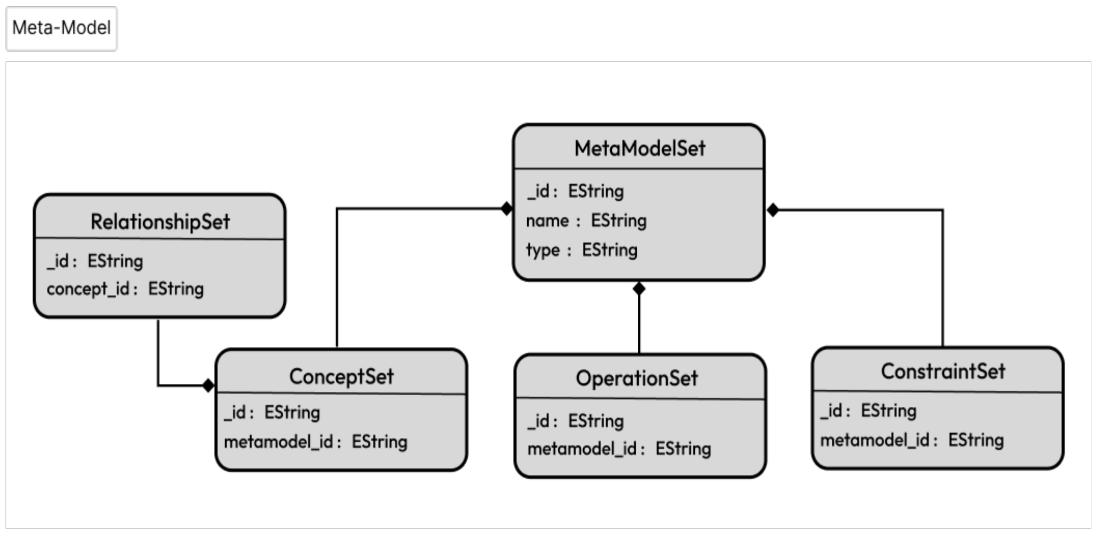


Figure 3.12: General Meta-model for Modeling Languages

3.5.1 Meta-model parts

User

The user is the main engine of this online platform by giving the Model which Source or Target, and engage with the platform to understand the intricacies of model transformation and its practical applications. They leverage the provided examples as a source of knowledge and inspiration, learning different techniques and approaches. Users also act as problem solvers, identifying specific challenges where model transformation can be applied to enhance existing models or create new ones. They define the desired target models, specifying the modifications and adaptations they seek. By actively participating in the platform, users contribute by providing their source models, allowing others to learn from their work. They actively interact with the examples, experimenting with different transformation techniques and adapting the demonstrated solutions to their own use cases. Additionally, users collaborate with fellow platform members, forming a community where they can share experiences, exchange ideas, and provide support. Their engagement and feedback help in improving the platform's functionality, ensuring its continuous growth as a valuable resource for the broader community interested in model transformation.

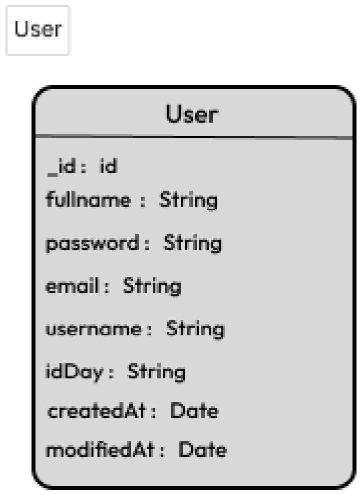


Figure 3.13: User part

Model Transformation

Model transformation is a crucial step in the process of creating a meta-model that is composed of three elements: transformation rules, heuristics, and criteria.

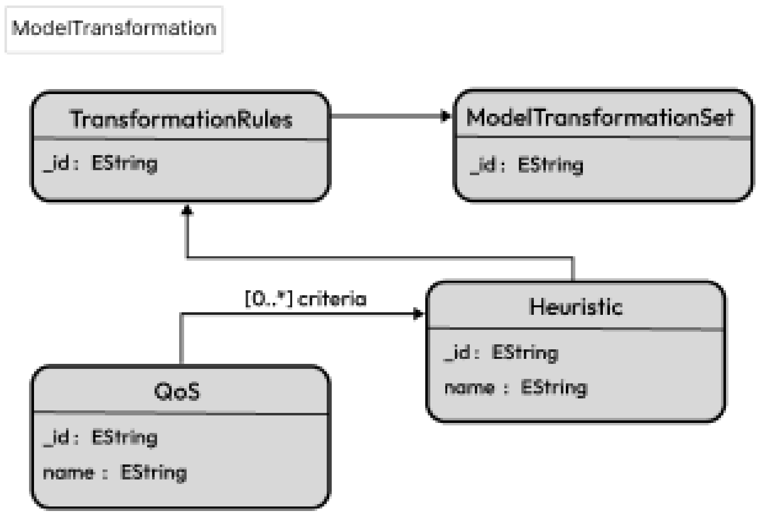


Figure 3.14: Model Transformation Part

Figure 3.14 shows the transformation rules that define how the input model (Source Model) should be transformed into the output model (Target Model), while the heuristics guide the transformation process by providing additional knowledge and insight. The criteria are used to evaluate the quality of the transformed model and ensure that it meets the intended requirements and specifications.

Heuristic

QoS attributes and heuristics play a crucial role in the context of model transformation, a process that involves converting a source model into a target model while preserving specific properties or characteristics. QoS (Quality of Service) attributes represent the desired qualities or performance metrics that need to be maintained or improved during the transformation process. These attributes can include reliability, scalability, availability, performance, security, and maintainability, among others. Heuristics, on the other hand, refer to the guidelines or strategies used to guide the model transformation process and achieve the desired QoS attributes. For example, when transforming a software architecture model, a heuristic may be to prioritize scalability by identifying components that can be distributed across multiple nodes. Another heuristic could be to optimize performance by identifying and eliminating bottlenecks or redundant operations. By employing appropriate heuristics, model transformation can effectively address the desired QoS attributes and ensure the resulting target model meets the specified quality requirements.

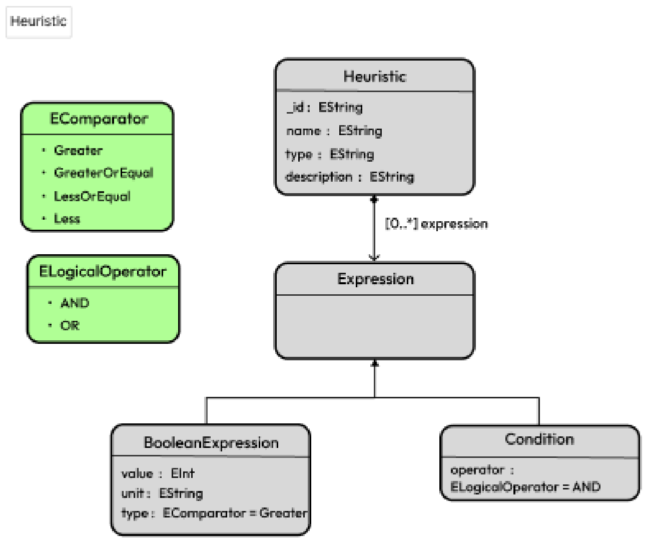


Figure 3.15: Heuristic part

Source and Target model

The source model represents the system or domain that is being modeled and typically conforms to a particular modeling language or format. The source model may be created manually or generated automatically from other sources, such as code or data.

The target model represents the desired output of the transformation process and may be in a different modeling language or format than the source model. The target model may be used for a variety of purposes, such as code generation, simulation, or analysis.

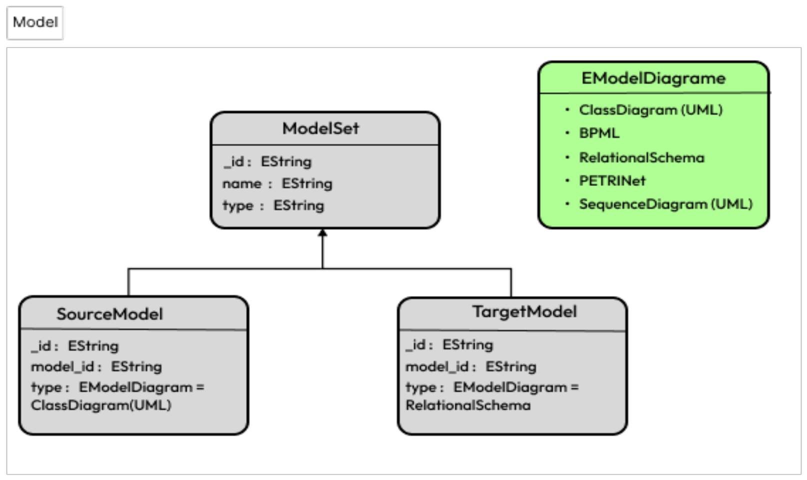


Figure 3.16: Source and Target Model Part

3.5.2 Gathering meta-model parts

After thoroughly defining the individual parts illustrated inFigure 3.13, Figure 3.15, Figure 3.16, Figure 3.14, and Figure 3.6 , we can now draw a comprehensive conclusion. By integrating and connecting these parts, we have successfully developed an extensible, adaptive, flexible, and general meta-model that can accommodate the characteristics of various modeling languages. The final result, depicted in Figure 3.17 , represents the culmination of our efforts to create a unified framework.

Figure 3.17 serves as a visual representation of the interconnectedness of the different parts and their relationships within the meta-model. It encapsulates the extensibility, adaptability, flexibility, and generality that we aimed to achieve. The interconnectedness of these parts enables the meta-model to capture the essential elements and structures found in diverse modeling languages, accommodating their unique characteristics and requirements.

With this meta-model, we provide a powerful tool that can be utilized across different modeling languages, facilitating a more comprehensive and unified approach to modeling. It allows designers, researchers, and practitioners to analyze, compare, and understand the fundamental components and relationships within modeling languages in a systematic and structured manner.

Moreover, the extensibility and flexibility of the meta-model ensure that it can be expanded upon and adapted to future advancements in modeling languages. It provides a solid foundation for incorporating new concepts, accommodating domain-specific requirements, and evolving with the changing needs of the modeling community.

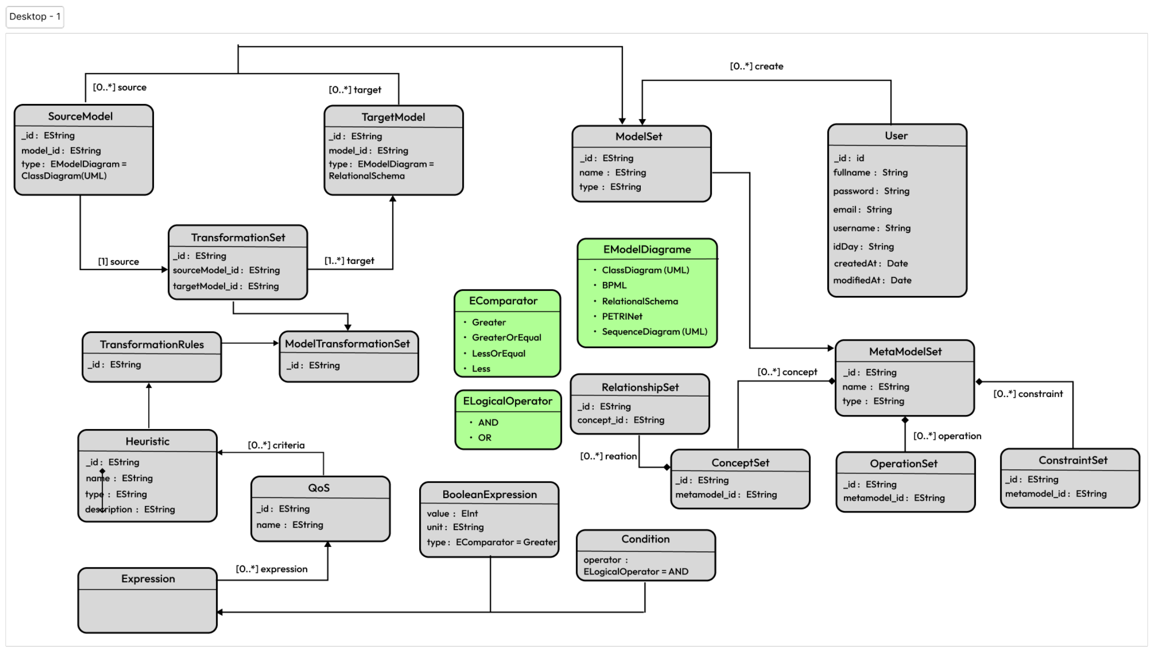


Figure 3.17: Gathering the meta-model parts to build the final form

3.6 Conclusion

In this chapter, we have analyzed five different modeling languages and used this analysis to deduce the components of a meta-model. We have also described the process of building a meta-model from scratch, which involves defining the abstract syntax, concrete syntax, and semantics of the meta-model.

Our analysis of the five modeling languages revealed common patterns and structures that are shared across different modeling domains. By identifying these patterns and structures, we were able to define a set of core components that are essential to any meta-model. These components include elements, relationships, attributes, constraints, and operations.

The process of building a meta-model from scratch involves defining the abstract syntax, which specifies the basic building blocks of the modeling language, such as classes and associations. The concrete syntax, which specifies how the modeling language is represented visually or textually, and the semantics, which defines the meaning and behavior of the modeling language, are also defined.

Building a meta-model from scratch is a complex and challenging task that requires a deep understanding of the domain being modeled and the principles of model-driven engineering. However, by following a rigorous and systematic approach, it is possible to create a meta-model that accurately represents the desired system or domain and can be used to generate high-quality models and code.

In conclusion, the process of analyzing multiple modeling languages and building a meta-model from scratch is an important step in the development of model-driven engineering tools and approaches. By understanding the common patterns and structures that underlie different modeling languages, it is possible to create meta-models that are flexible, extensible, and can be used in a wide range of modeling domains and contexts.

Chapter 4

Implementation of our application

4.1 Introduction

Building a website with the goal of achieving model transformation through online examples opens up exciting possibilities in the realm of Model-Driven Engineering (MDE). The integration of website development and online examples as a means to enable and empower users to experience model transformations firsthand. By leveraging the interactive nature of websites and the power of online examples, users can gain a deeper understanding of model transformation concepts and techniques in a practical and accessible way.

4.2 Client-server architecture

In the computing world today, client-server system has become so popular because it is being used virtually every day for different applications. Some of the standardized protocols that client and servers use to communicate with themselves include: File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP) and Hypertext Transfer Protocol (HTTP). Thus, Client-server system can be define as a software architecture made up of both the client and server, whereby the clients always send requests while the server responds to the requests sent[3]. Client-server provides an inter-process communication because it involves the exchange of data from both the client and server whereby each of them performs different functions [28]

Client-Side (Frontend):

The client-side, or frontend, is responsible for the user interface and presentation layer that users interact with. It includes HTML, CSS, and JavaScript code that is rendered in the user's web browser. Common frontend frameworks and libraries include React, Angular, or Vue.js. The frontend interacts with the backend through APIs to retrieve and manipulate data.

Server-Side (Backend)

The server-side, or backend, handles the logic and processing of data. It consists of web servers, application servers, and databases. The backend is responsible for receiving requests from the frontend, performing operations, and sending back the appropriate responses. Backend technologies can include Node.js, Python (with frameworks like Django or Flask), Ruby (with Ruby on Rails), or Java (with Spring framework).

Databases:

Data generated by the website is typically stored in a database. Relational databases like MySQL, PostgreSQL, or Oracle are commonly used for structured data. NoSQL databases like MongoDB or Cassandra are suitable for unstructured or semi-structured data. The choice of database depends on the specific requirements of the website.

4.3 Technologies used

4.3.1 Front-end

ReactJS

React.js is a popular JavaScript library used for building user interfaces. It simplifies the process of creating interactive and dynamic web applications. With React.js, developers can break down the user interface into reusable components, each responsible for a specific part of the application. These components can be combined to build complex UIs, making it easier to manage and maintain code. React.js uses a virtual DOM (Document Object Model) which efficiently updates and renders only the necessary parts of the UI, resulting in faster performance. React.js also supports a declarative syntax, allowing developers to describe how the UI should look based on the application's state. This helps in building robust and scalable applications with less code and fewer bugs.[29]

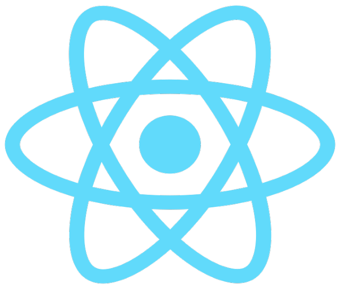


Figure 4.1: ReactJs

4.3.2 Back-end

NodeJs

Node.js is a JavaScript runtime environment that allows developers to run JavaScript code outside of a web browser. It enables the execution of JavaScript on the server-side, making it possible to build fast and scalable web applications. Node.js is built on the V8 JavaScript engine, which provides high-performance execution. One of the key advantages of Node.js is its non-blocking, event-driven architecture, which allows it to handle a large number of simultaneous connections efficiently. This makes it suitable for building real-time applications, APIs, and microservices. Node.js has a vast ecosystem of modules and packages available through its package manager, npm, which makes it easy to integrate third-party libraries into projects. Its simplicity and flexibility have made Node.js a popular choice for server-side development.[30]

Figure 4.2: ExpressJs

ExpressJs

Express.js is a minimalistic and flexible web application framework for Node.js. It provides a straightforward and easy-to-use set of features for building web servers and APIs. With Express.js, developers can handle HTTP requests, define routes, and manage middleware for processing incoming and outgoing data. It simplifies the process of creating web applications by offering a clean and intuitive API. Express.js also allows for the integration of additional middleware and modules to extend its functionality. It is known for its simplicity and lightweight nature, making it a popular choice for building small to medium-sized web applications. Whether you need to create a simple API or a full-fledged web server, Express.js provides a solid foundation for developing web applications with Node.js.[31]

Figure 4.3: ExpressJs

4.3.3 DataBase

MongoBD

MongoDB is a popular and flexible NoSQL database that provides a scalable and high-performance solution for storing and managing data. It is designed to handle large volumes of structured, semi-structured, and unstructured data across distributed systems. MongoDB uses a document-oriented data model, where data is stored in flexible JSON-like documents instead of traditional rows and columns. This allows for dynamic and schema-less data structures, making it easier to adapt to changing application requirements. MongoDB supports powerful query capabilities, indexing, and aggregation for efficient data retrieval and analysis. It also offers features like automatic sharding and replication for horizontal scalability and high availability. With its ease of use, scalability, and versatility, MongoDB is widely used in a range of applications, including web and mobile development, real-time analytics, and content management systems.[30]

Figure 4.4: MongoDB

4.4 Application Working

4.4.1 MVC design paradigm

Our website is based on model view controller or M-V-C design paradigm is popular and fundamental to the user interface development, not only in web apps but in front-end applications running on any platform. DOM represents physical View in case of web-applications. The DOM is made via a HTML template which itself is fetched from a different file, script block or a precompiled template function. The textual template is given life as a DOM by the View entity itself. It plays a key role of handling the Events and manipulating the document object model tree as a part of its life cycle. A view can only be useful if and only if it makes the user interaction possible as well as display the required data. Data is an entity that is brought from some

Data-Store, which could be a Database ,a web service or a Local Store. Frameworks provide a way to bind view to the data store in order to make sure that changes made to the database are automatically reflected back. This process of automatic data updates pushing is commonly referred to as DataBinding. There are many application program interfaces or API's which make this process merely a cakewalk. As shown in Figure 4.5, the M-V-C paradigm is completed by the C component i.e. the Controller which engages the rest two components i.e. the model and the view and enables the data model flow into the View and user events out of View, eventually leading to changes in the Model. [29]

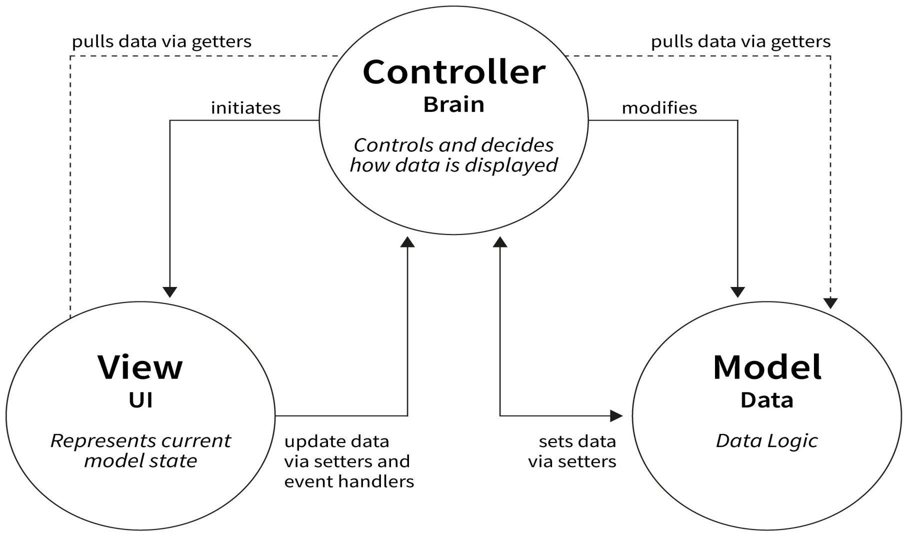
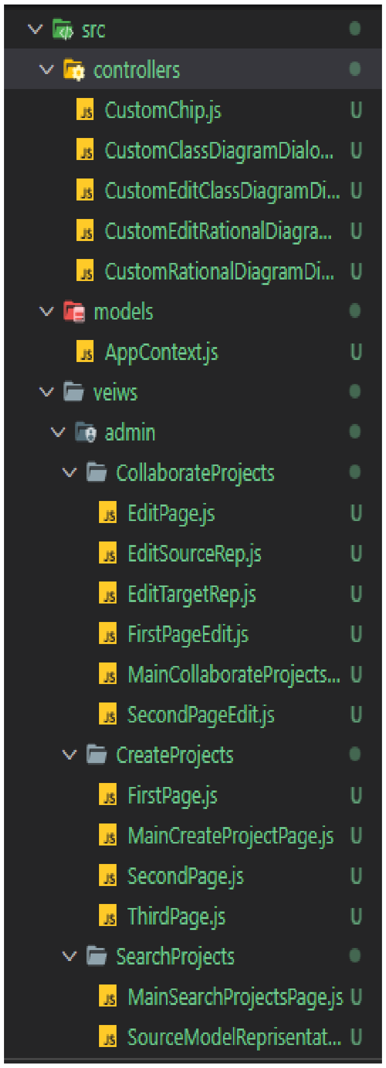


Figure 4.5: MVC Architecture Pattern

File Structure



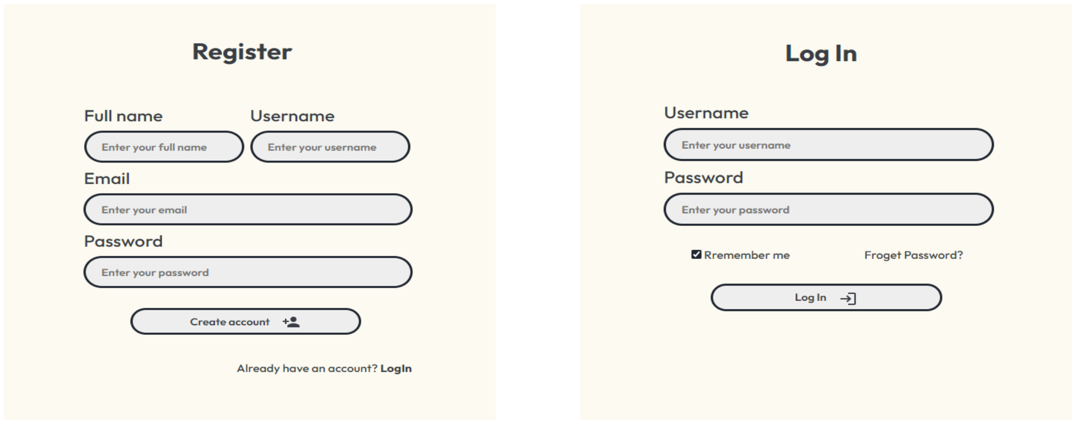
```
src
  controllers
    CustomChip.js                     U
    CustomClassDiagramDialo...        U
    CustomEditClassDiagramDi...       U
    CustomEditRationalDiagra...       U
    CustomRationalDiagramDi...        U
  models
    AppContext.js                     U
  veiws
    admin
      CollaborateProjects
        EditPage.js                   U
        EditSourceRep.js              U
        EditTargetRep.js              U
        FirstPageEdit.js              U
        MainCollaborateProjects...    U
        SecondPageEdit.js             U
      CreateProjects
        FirstPage.js                  U
        MainCreateProjectPage.js      U
        SecondPage.js                 U
        ThirdPage.js                  U
      SearchProjects
        MainSearchProjectsPage.js     U
        SourceModelReprisentat...     U
```

Figure 4.6: File Structure based on MVC

4.4.2 Illustrate the application

Register/Login

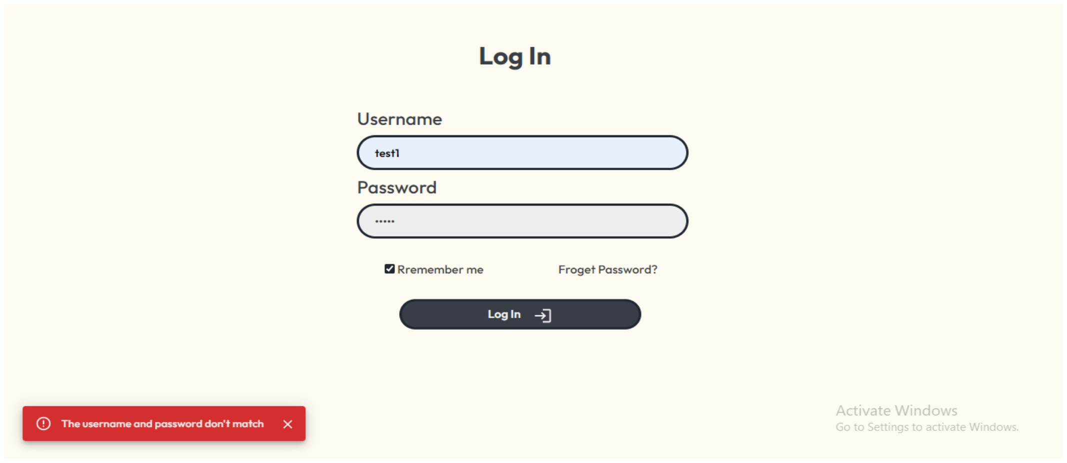
User login registration is a fundamental process that allows individuals to create an account and access various services or platforms. It involves providing necessary information such as a username, password, and often additional details like email.



(a) Register (b) Login

Figure 4.7: Register/Login page

If the user makes mistakes in the password, the system informs him in the form of alert



test1

•••••

The username and password don't match

Figure 4.8: Login alert

Dashboard

The user dashboard offers three primary options to users: creating meta-models, searching meta-models, and collaborating.

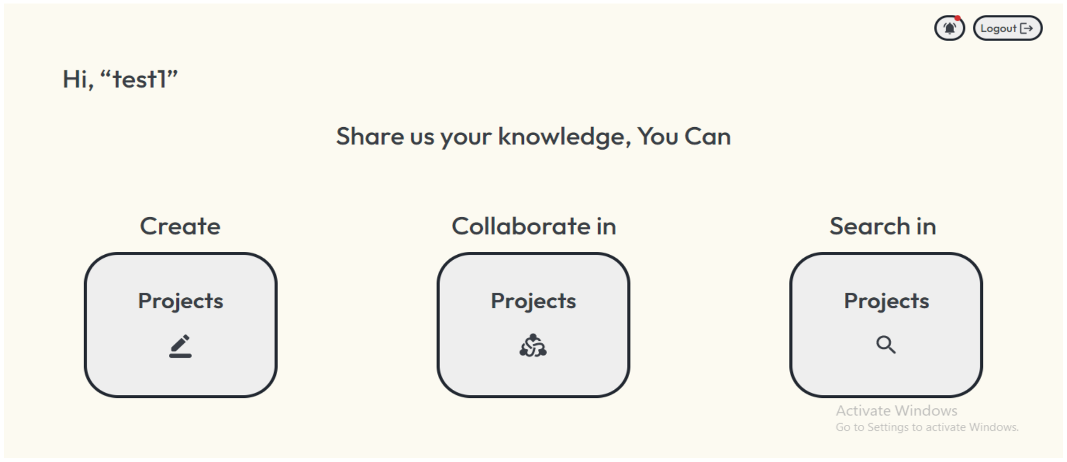
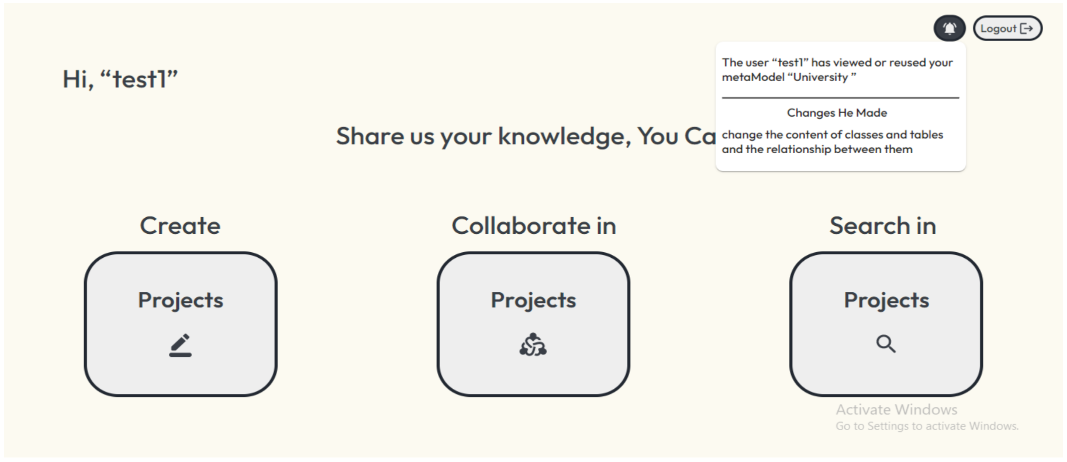


Figure 4.9: User Dashboard

The notification icon ensures that users stay informed about any updates or actions related to their meta models. By having this visibility, users can promptly review and respond to changes, stay updated on the activity surrounding their models, and maintain an active and engaged presence within the platform.


The user "test1" has viewed or reused your metaModel "University "

Changes He Made

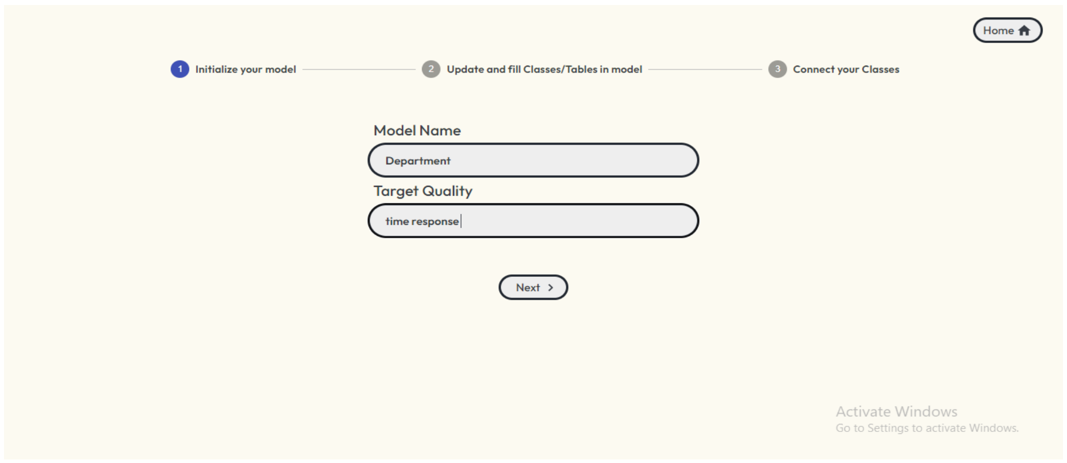
change the content of classes and tables and the relationship between them

Figure 4.10: Dashboard notification

Create Meta-model

This option allows users to build their own meta-models. Meta models are high-level models that help to organize and interpret complex systems or datasets. By creating meta-models, users can analyze and understand intricate relationships, patterns, and dependencies within their data. This process often involves defining variables, establishing connections, and specifying rules or algorithms. Creating meta-models empowers users to gain valuable insights and make informed decisions based on their unique data and objectives.

First step : Initialize the model by setting his name and the target QoS.

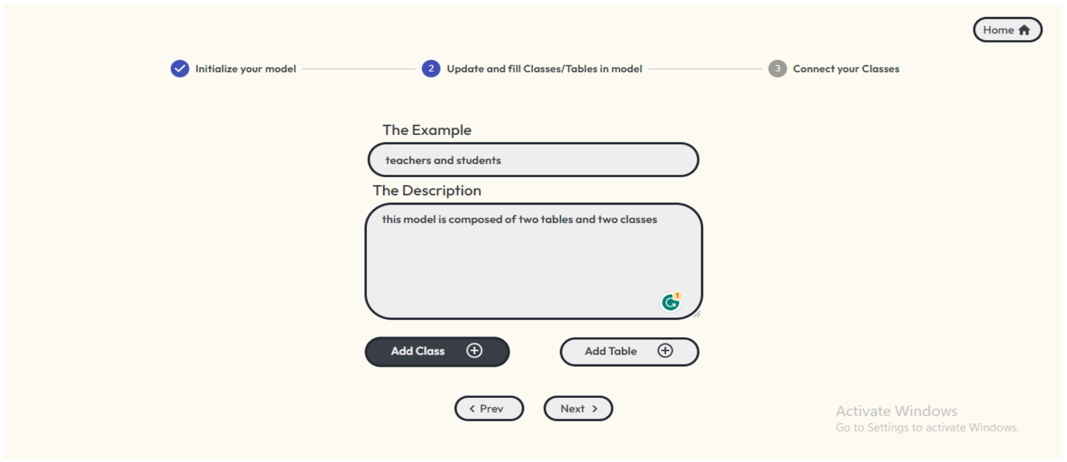

Home
1 Initialize your model
2 Update and fill Classes/Tables in model
3 Connect your Classes
Next
Activate Windows
Go to Settings to activate Windows.

Model Name: Department

Target Quality: time response

Figure 4.11: Initialize model

Second step : Update the tables/Classes and describe the model.



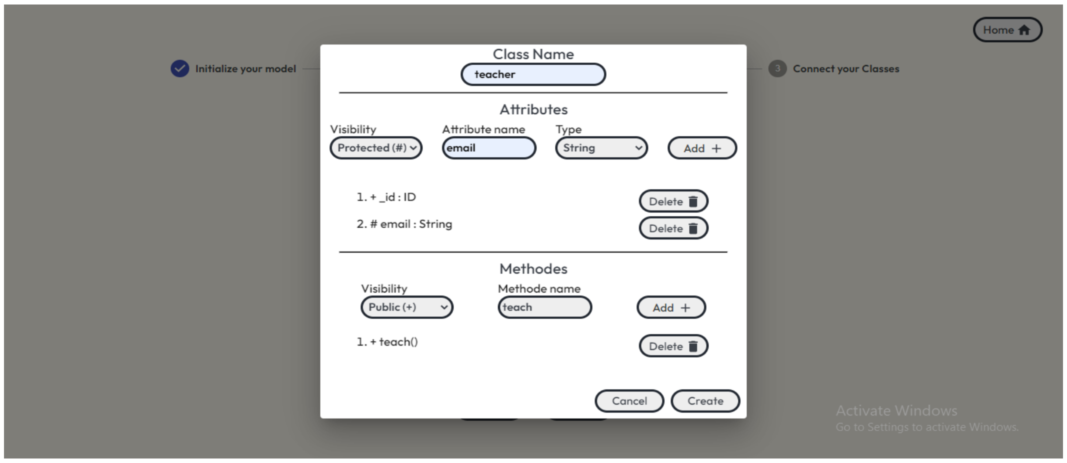
The Example

teachers and students

The Description

this model is composed of two tables and two classes

Figure 4.12: Describe the model



Class Name: teacher

Attributes

Visibility: Protected (#) | Attribute name: email | Type: String

1. + _id : ID
2. # email : String

Methodes

Visibility: Public (+) | Methode name: teach

1. + teach()

Figure 4.13: Create Class

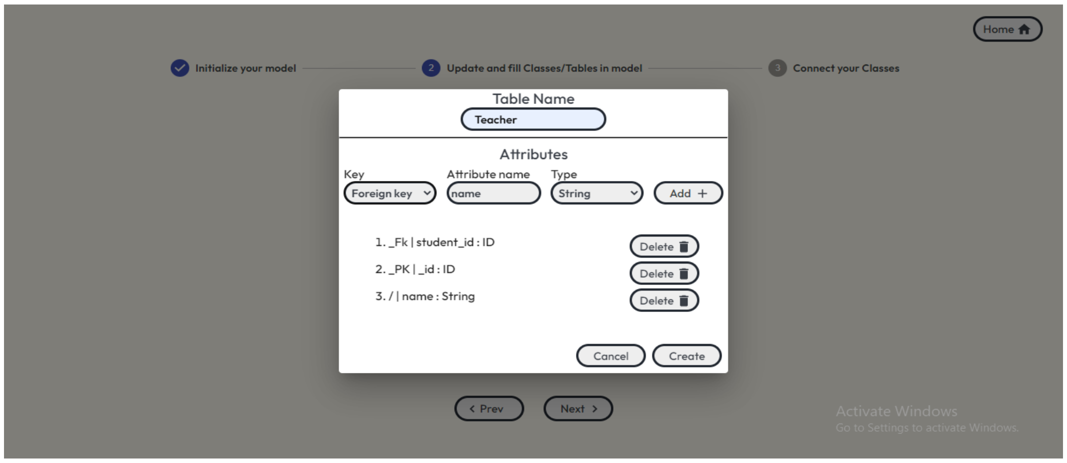


Table Name

Teacher

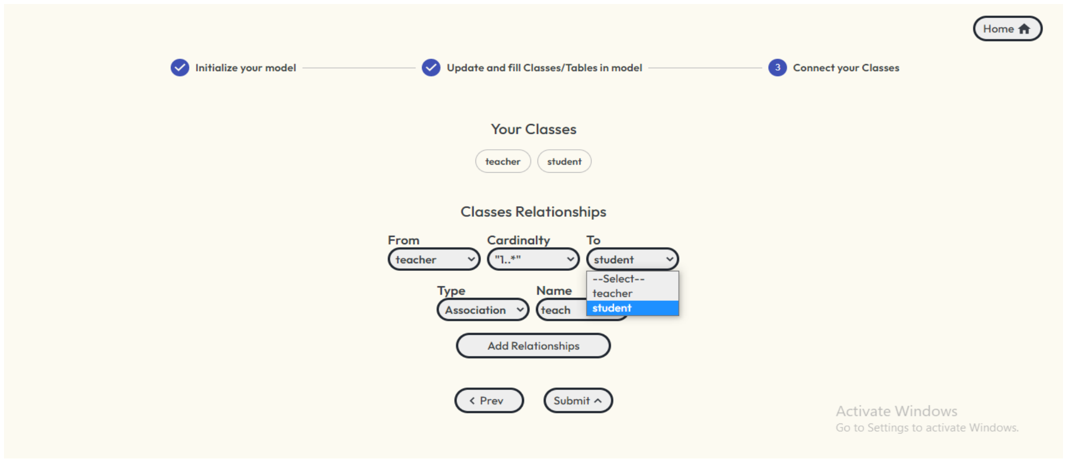
Attributes

Key: Foreign key | Attribute name: name | Type: String | Add +

1. _Fk | student_id : ID
2. _PK | _id : ID
3. / | name : String

Figure 4.14: Create Table

Third step : Connect all the Classes the user creates with relationships and corresponding cardinality.



Your Classes

teacher student

Classes Relationships

From: teacher | Cardinalty: "1..*" | To: student

Type: Association | Name: teach

--Select--
teacher
student

Figure 4.15: Create Relationships between classes

If you connect all your classes the system informs you that You have related all the Classes.

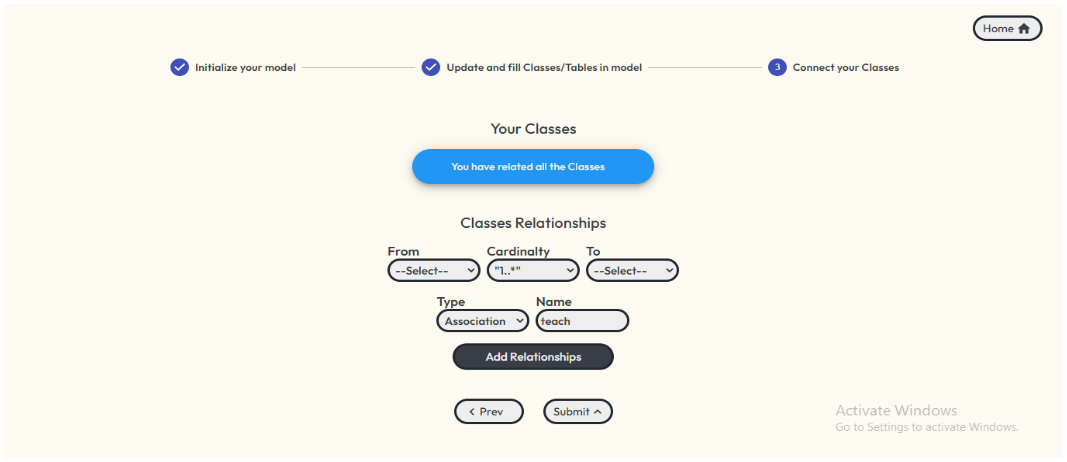


Figure 4.16: Relationships inform alert

If you don't connect all your classes and click on submit button an alert will appear to inform you

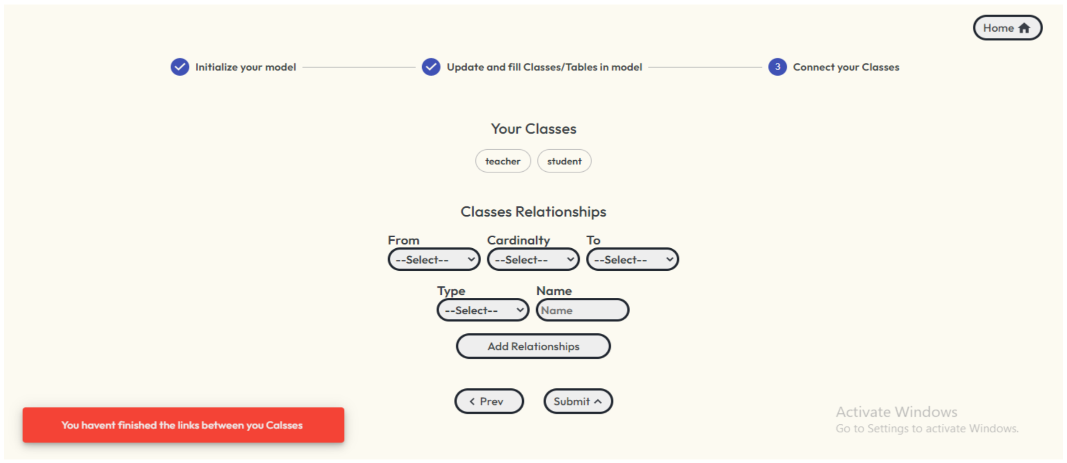
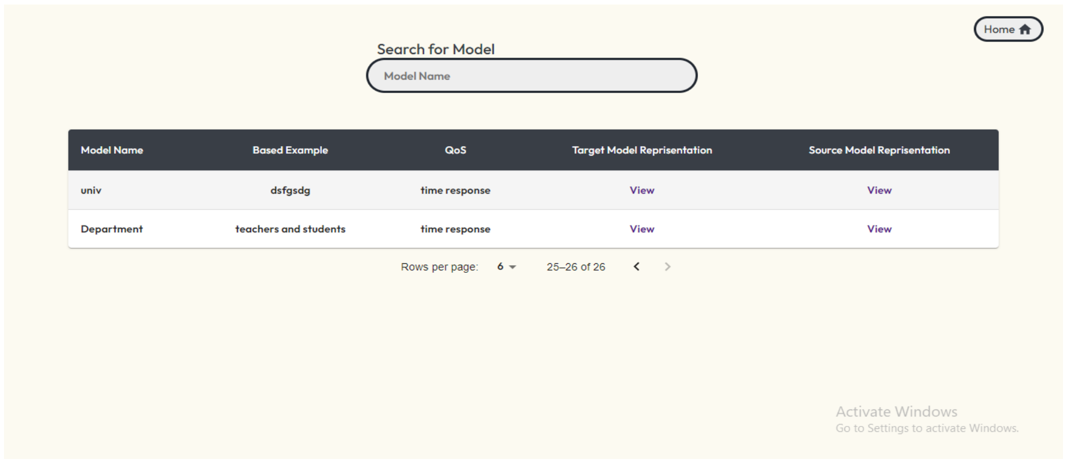


Figure 4.17: Relationships error alert

Searcher Meta-model

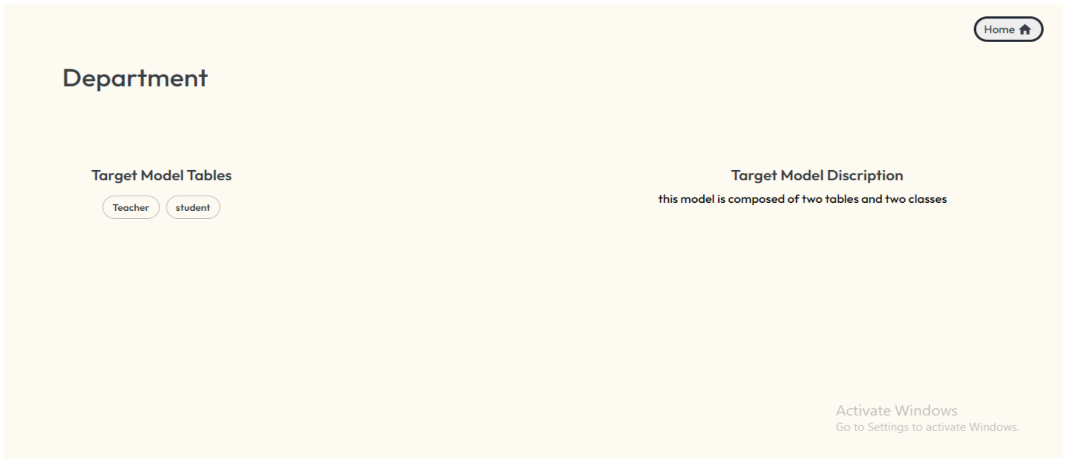
This option enables users to explore existing meta-models created by others. Users can leverage the power of collective knowledge and expertise by searching for meta-models relevant to their domain or interests. By accessing pre-built meta models, users can save time and effort in constructing models from scratch. They can study and adapt existing models to their specific needs or integrate them into their own workflows, accelerating their analysis or problem-solving processes.



Model Name	Based Example	QoS	Target Model Reprisentation	Source Model Reprisentation
univ	dsfgsdg	time response	View	View
Department	teachers and students	time response	View	View

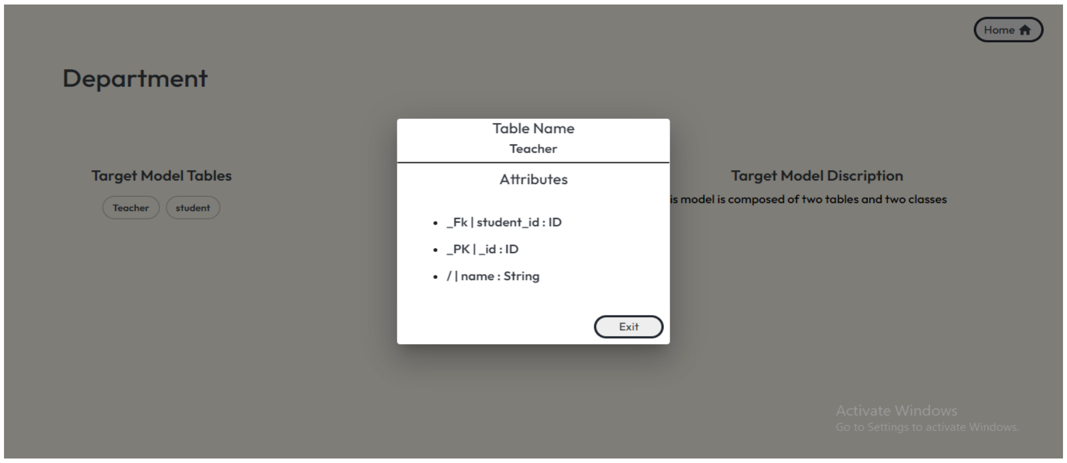
Figure 4.18: Search page

Target Model Representation



this model is composed of two tables and two classes

Figure 4.19: Target Model Representation page



Department

Target Model Tables

Teacher student

Table Name
Teacher

Attributes

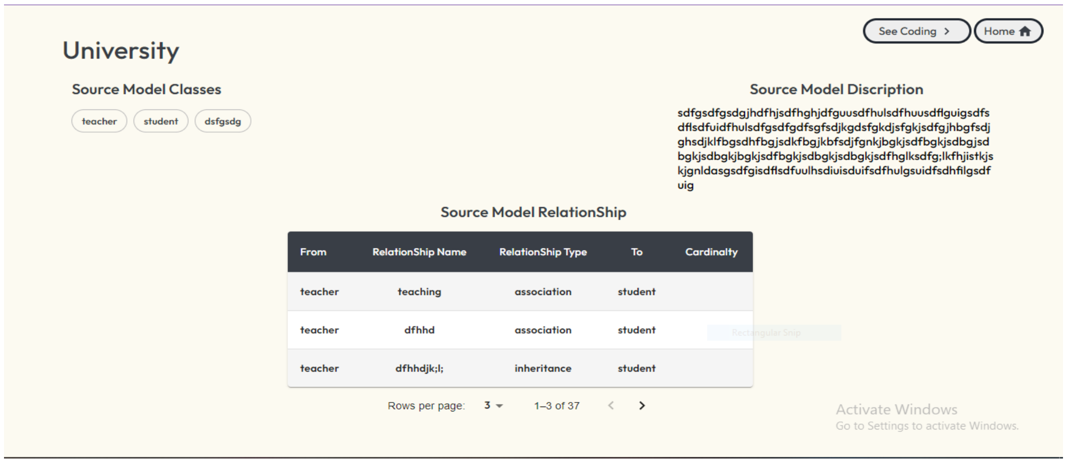
- _Fk | student_id : ID
- _PK | _id : ID
- / | name : String

Target Model Discription

is model is composed of two tables and two classes

Figure 4.20: Table Representation

Source Model Representation



University

Source Model Classes

teacher student dsfgsdg

Source Model Discription

sdfgsdfgsdgjhdfhjsdfhghjdfguusdfhulsdfhuusdflguigsdfs
dflsdfuidfhulsdfgsdfgdfsgfsdjkgdsfgkdjsfgkjsdfgjhbgfsdj
ghsdjklfbgsdhfbgjsdkfbgjkbfsdjfgnkjbgkjsdfbgkjsdbgjsd
bgkjsdbgkjbgkjsdfbgkjsdbgkjsdbgkjsdfhglksdfg;lkfhjistkjs
kjgnldasgsdfgisdflsdfuulhsdiuisduifsdfhulgsuidfsdhfilgsdf
uig

Source Model RelationShip

From	RelationShip Name	RelationShip Type	To	Cardinalty
teacher	teaching	association	student	
teacher	dfhhd	association	student	
teacher	dfhhdjk;l;	inheritance	student	

Rows per page: 3 1–3 of 37

Figure 4.21: Source Model Representation page

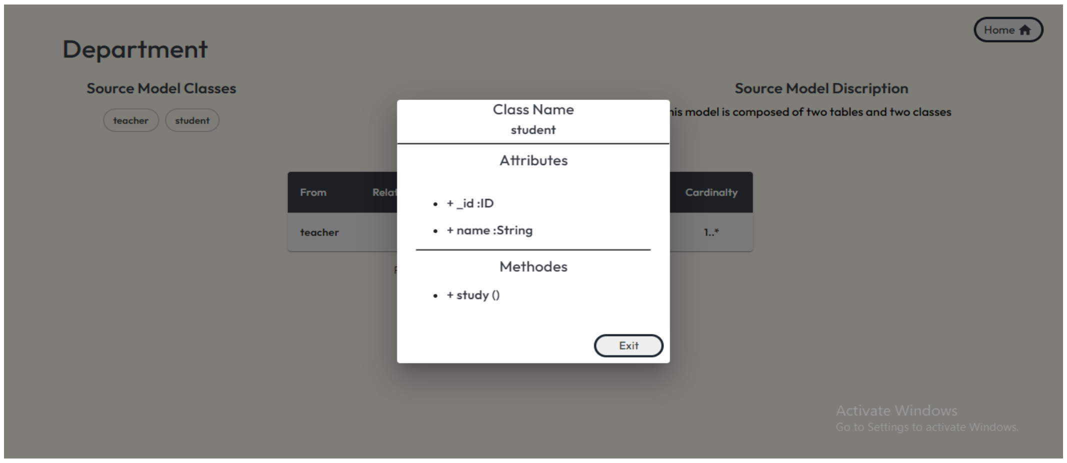
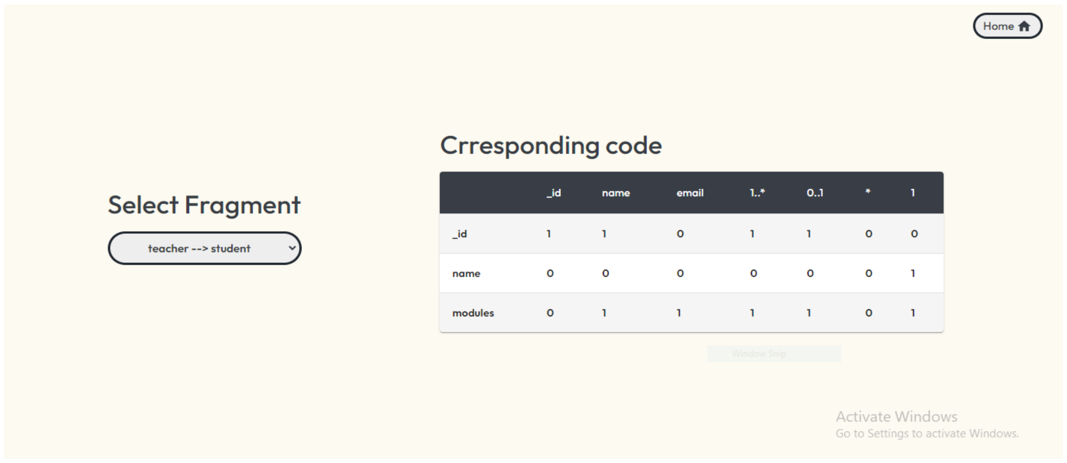


Figure 4.22: Class Representation

When the user clicks the ”See Code” button, they will be redirected to the coding model page. On this page, the user will have the option to select which fragment of the model they would like to code. The platform will provide a user-friendly interface where the user can make their selection.

Once the user has chosen the desired fragment, the platform will regenerate the table based on the selected fragment. The table will be dynamically generated to display the relevant attributes and information related to the chosen fragment. This allows the user to focus specifically on the code for that particular section of the model.

The platform aims to provide a seamless coding experience, enabling users to easily navigate through different model fragments and efficiently work on coding tasks. By offering this level of flexibility and customization, users can streamline their coding process and efficiently handle different parts of the model as required.



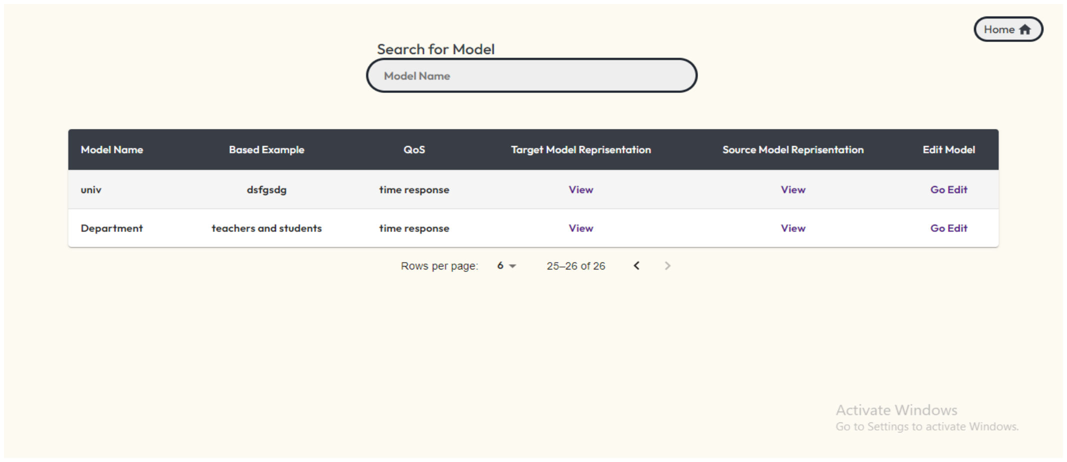
Crresponding code

	_id	name	email	1..*	0..1	*	1
_id	1	1	0	1	1	0	0
name	0	0	0	0	0	0	1
modules	0	1	1	1	1	0	1

Figure 4.23: coding a fragment from the model

Collaborate in Meta-model

Collaboration is a crucial aspect of the user dashboard, fostering teamwork and knowledge sharing. Users can collaborate with other individuals or teams by sharing their meta models, seeking feedback, or engaging in joint projects. Collaboration encourages diverse perspectives, promotes innovation, and enhances the quality of models through collective efforts. Users can exchange ideas, refine models, and collectively tackle complex problems, leading to more robust and accurate results.

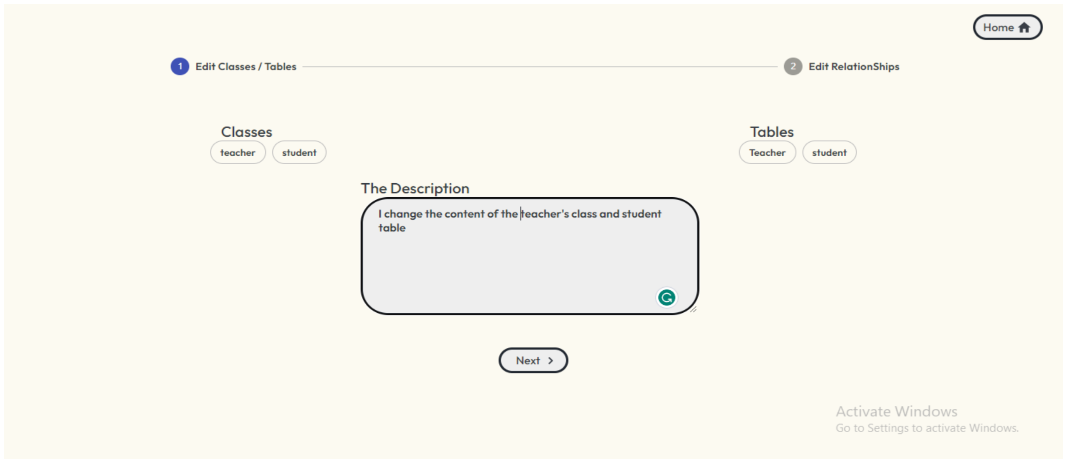


Model Name	Based Example	QoS	Target Model Reprisentation	Source Model Reprisentation	Edit Model
univ	dsfgsdg	time response	View	View	Go Edit
Department	teachers and students	time response	View	View	Go Edit

Rows per page: 6 25–26 of 26

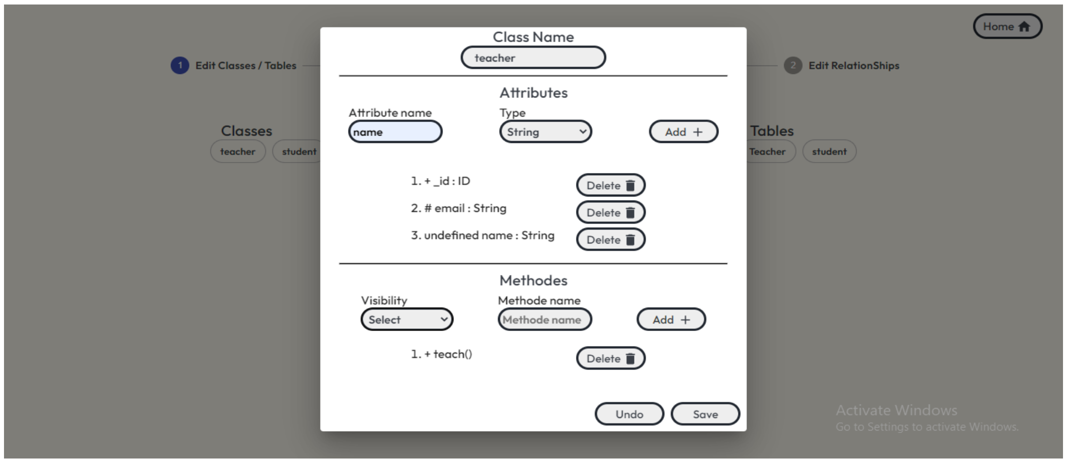
Figure 4.24: Collaboration page

First step : After the user choose the model for reuse the application creates a clone of the model classes and table that allows the user to manipulate table class name, attributes, methods, and describe the changes .



I change the content of the teacher's class and student table

Figure 4.25: Edit page



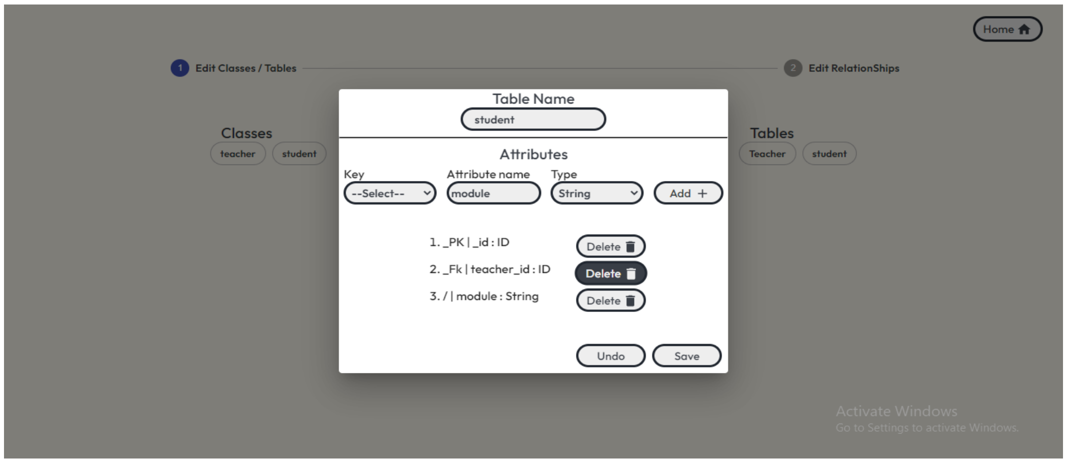
Class Name: teacher

Attribute name: name
Type: String

1. + _id : ID
2. # email : String
3. undefined name : String

1. + teach()

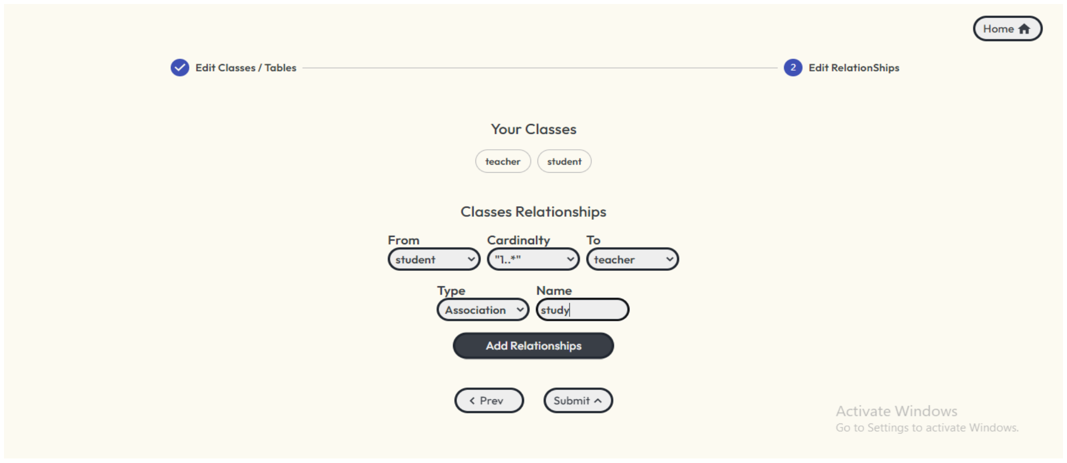
Figure 4.26: Class manipulation



1. _PK | _id : ID
2. _Fk | teacher_id : ID
3. / | module : String

Figure 4.27: Table manipulation

Second step : The second step focus on the manipulation of the relationship between classes.



From: student
Cardinalty: "1..*"
To: teacher
Type: Association
Name: study

Figure 4.28: Relationships manipulation

4.5 Conclusion

In this chapter, we have provided an in-depth overview of the technical requirements necessary for the development of our system. We identified and discussed the key components and functionalities that needed to be implemented to achieve our solution.

Following the discussion of the technical requirements, we proceeded to the realization stage of our solution. This involved the actual implementation of each component of the system. We meticulously developed and integrated each component, ensuring that they functioned harmoniously to meet the desired objectives of the system.

Towards the end of the chapter, we presented the technical architecture of our solution. We outlined the overall structure and organization of the system, highlighting the relationships and interactions between its various components. This architectural overview provided a clear understanding of how the different parts of the system worked together to deliver the desired functionality. we delved into the functionalities and interactions of the different components of the system. We explained how each component contributed to the overall system's functionality, clarifying their roles and responsibilities. This comprehensive understanding of component functionalities allowed us to demonstrate the cohesive and integrated nature of our solution.

By providing a comprehensive overview of the technical requirements, the implementation process, the system's architecture, user interactions, and component functionalities, we have presented a detailed and holistic perspective of our system. This chapter serves as a foundation for further exploration, evaluation, and utilization of our solution, enabling users and stakeholders to gain a thorough understanding of its capabilities and potential benefits.

Chapter 5

General conclusion

In this section, we will present the general conclusion and future work.

5.1 Conclusion

In conclusion, the development of the online platform has demonstrated its effectiveness and efficiency in real-world scenarios, enhancing the practicality and usability of model transformation by example. By providing creation, modification, and persistence capabilities, the platform has facilitated a more interactive and efficient model transformation process. Modelers can now iterate and refine their transformations with greater effectiveness, leading to improved overall productivity.

This thesis has made significant advancements in the field of model-driven engineering and dataset creation. The developed framework empowers researchers, practitioners, and the broader computer science community to drive innovation and advance the field through standardized, efficient, and collaborative dataset creation processes. It has improved the efficiency of model transformation, enhanced the quality of generated datasets, and fostered a collaborative and knowledge-sharing community. The findings of this research have practical implications and provide a solid foundation for future studies in the domain of model-driven engineering and online persistence model transformation.

The collaborative features of the framework have promoted teamwork and knowledge sharing among modelers. The ability to collaborate, reuse models, and share datasets has fostered a collaborative community within the computer science field. Concurrent access, conflict resolution, and collaborative editing have enabled modelers to work together, benefiting the entire model-driven engineering community.

5.2 Perspectives

The online platform proposed in this thesis offers several perspectives and potential benefits to the community of computer science. These perspectives can be viewed from different angles, including researchers, practitioners, and the broader community. Here are some perspectives on the online platform:

1. **More extensible and adaptive platform :** is archived by the increase in the number of modeling languages in the conception of the online platform by not making it limited to two modeling languages.
2. **Community Building and Networking:** The online platform fosters a sense of community and networking within the computer science domain. Users can connect with like-minded individuals, engage in discussions, and establish professional relationships. This perspective highlights the potential for networking opportunities, collaborations, and the development of a strong community that collectively contributes to dataset creation and other areas of research.
3. **Iterative Improvement and Feedback:** The platform encourages iterative improvement and feedback loops. Users can share their work, receive feedback from peers, and continuously refine their models and datasets. This perspective underscores the value of ongoing improvement, learning, and growth within the community, leading to higher-quality datasets and more robust research outcomes.
4. **Impact on Computer Science Community:** The online platform has the potential to contribute to the wider computer science community. By fostering collaboration, promoting best practices, and sharing datasets, the platform can have a lasting impact on the advancement of computer science research. It can serve as a catalyst for new discoveries, innovative approaches, and cross-disciplinary collaborations, benefiting the broader community of researchers, practitioners, and students.

Bibliography

[1] Douglas C Schmidt et al. Model-driven engineering. *Computer-IEEE Computer Society-*, 39(2):25, 2006.

[2] Jeff Rothenberg, Lawrence E Widman, Kenneth A Loparo, and Norman R Nielsen. The nature of modeling. *in Artificial Intelligence, Simulation and Modeling*, 1989.

[3] Anneke G Kleppe, Jos B Warmer, Jos Warmer, and Wim Bast. *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional, 2003.

[4] Frank Budinsky, Raymond Ellersick, David Steinberg, Timothy J Grose, and Ed Merks. *Eclipse modeling framework: a developer's guide*. Addison-Wesley Professional, 2004.

[5] JF Overbeek. Meta object facility (mof): investigation of the state of the art. Master's thesis, University of Twente, 2006.

[6] Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. *Electronic notes in theoretical computer science*, 152:125–142, 2006.

[7] Gerti Kappel, Philip Langer, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. Model transformation by-example: a survey of the first wave. *Conceptual Modelling and Its Theoretical Foundations: Essays Dedicated to Bernhard Thalheim on the Occasion of His 60th Birthday*, pages 197–215, 2012.

[8] Krzysztof Czarnecki and Simon Helsen. Classification of model transformation approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, volume 45, pages 1–17. USA, 2003.

[9] Jeffrey Gene Gray. *Aspect-oriented domain-specific modeling: A generative approach using a metaweaver framework*. Vanderbilt University, 2002.

[10] Mel Ó Cinnéide and Paddy Nixon. Automated software evolution towards design patterns. In *Proceedings of the 4th international workshop on Principles of software evolution*, pages 162–165, 2001.

[11] Daniel Conrad Halbert. *Programming by example*. University of California, Berkeley, 1984.

[12] Dániel Varró. Model transformation by example. In *Model Driven Engineering Languages and Systems: 9th International Conference, MoDELS 2006, Genova, Italy, October 1-6, 2006. Proceedings 9*, pages 410–424. Springer, 2006.

[13] Manuel Wimmer, Michael Strommer, Horst Kargl, and Gerhard Kramler. Towards model transformation generation by-example. In *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, pages 285b–285b. IEEE, 2007.

[14] Michael Strommer and Manuel Wimmer. A framework for model transformation by-example: Concepts and tool support. In *Objects, Components, Models and Patterns: 46th International Conference, TOOLS EUROPE 2008, Zurich, Switzerland, June 30-July 4, 2008. Proceedings 46*, pages 372–391. Springer, 2008.

[15] Marouane Kessentini, Manuel Wimmer, Houari Sahraoui, and Mounir Boukadoum. Generating transformation rules from examples for behavioral models. In *Proceedings of the Second International Workshop on Behaviour Modelling: Foundation and Applications*, pages 1–7, 2010.

[16] Zoltán Balogh and Dániel Varró. Model transformation by example using inductive logic programming. *Software & Systems Modeling*, 8(3):347–364, 2009.

[17] Iván García-Magariño, Jorge J Gómez-Sanz, and Rubén Fuentes-Fernández. Model transformations for improving multi-agent system development in ingenias. In *Agent-Oriented Software Engineering*

X: 10th International Workshop, AOSE 2009, Budapest, Hungary, May 11-12, 2009, Revised Selected Papers 10, pages 51–65. Springer, 2011.

[18] Xavier Dolques, Marianne Huchard, Clémentine Nebut, and Philippe Reitz. Learning transformation rules from transformation examples: An approach based on relational concept analysis. In *2010 14th IEEE International Enterprise Distributed Object Computing Conference Workshops*, pages 27–32. IEEE, 2010.

[19] Hajer Saada, Xavier Dolques, Marianne Huchard, Clémentine Nebut, and Houari Sahraoui. Generation of operational transformation rules from examples of model transformations. In *Model Driven Engineering Languages and Systems: 15th International Conference, MODELS 2012, Innsbruck, Austria, September 30–October 5, 2012. Proceedings 15*, pages 546–561. Springer, 2012.

[20] Martin Faunes, Houari Sahraoui, and Mounir Boukadoum. Generating model transformation rules from examples using an evolutionary algorithm. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 250–253, 2012.

[21] Islem Baki, Houari Sahraoui, Quentin Cobbaert, Philippe Masson, and Martin Faunes. Learning implicit and explicit control in model transformations by example. In *Model-Driven Engineering Languages and Systems: 17th International Conference, MODELS 2014, Valencia, Spain, September 28–October 3, 2014. Proceedings 17*, pages 636–652. Springer, 2014.

[22] Mirco Franzago, Davide Di Ruscio, Ivano Malavolta, and Henry Muccini. Collaborative model-driven software engineering: a classification framework and a research map. *IEEE Transactions on Software Engineering*, 44(12):1146–1175, 2017.

[23] Omar Alam, Jonathan Corley, Constantin Masson, and Eugene Syriani. Challenges for reuse in collaborative modeling environments. In *MoDELS (Workshops)*, pages 277–283, 2018.

[24] Hans-Erik Eriksson, Magnus Penker, Brian Lyons, and David Fado. *UML 2 toolkit*. John Wiley & Sons, 2003.

[25] Li Yang and Li Cao. The effect of mysql workbench in teaching entity-relationship diagram (erd) to relational schema mapping. *International Journal of Modern Education and Computer Science*, 8(7):1, 2016.

[26] Matthew Hause et al. The sysml modelling language. In *Fifteenth European Systems Engineering Conference*, volume 9, pages 1–12, 2006.

[27] Carl Adam Petri and Wolfgang Reisig. Petri net. *Scholarpedia*, 3(4):6477, 2008.

[28] S Kratky and C Reichenberger. Client/server development based on the apple event object model, 2013.

[29] Sanchit Aggarwal et al. Modern web-development using reactjs. *International Journal of Recent Research Aspects*, 5(1):133–137, 2018.

[30] Mithun Satheesh, Bruno Joseph D'mello, and Jason Krol. *Web development with MongoDB and NodeJs*. Packt Publishing Ltd, 2015.

[31] Azat Mardan. *Express. js Guide: The Comprehensive Book on Express. js*. Azat Mardan, 2014.