The Journal of Systems & Software

Microservice-based model-driven architecture for implementing modeling and model-transformation as a service

--Manuscript Draft--

Manuscript Number:	JSSOFTWARE-D-23-01192
Article Type:	VSI:MDE4SA
Keywords:	Model-Driven Engineering; Microservice; Model Transformation; Domain-Specific Language
Abstract:	Model-driven engineering (MDE) is a popular software engineering approach that emphasizes the use of models throughout the software development lifecycle. The goal of MDE is to reduce accidental complexity, and thereby leads to several benefits, including automatic code generation; however, there are several challenges that make it difficult to use in industry. These challenges include scalability issues, such as model size and complexity, and issues related to collaborative modeling and artifact reuse. In addition, the need remains for a clear-cut framework and methodology for applying MDE in an agile context. We propose a model-driven architecture that utilizes model-driven development (MDD) practices and service-oriented approaches through the microservice architecture. To this aim, we introduce the idea of multi-level modeling as a service and model transformation as a service and develop a model-driven platform (called MDDPlatform) to address issues related to reusability, scalability, and collaboration in software development. We have evaluated the proposed approach through a comparative analysis based on relevant metrics and criteria, and also through a case study that assessed the impact of the approach on relevant quality attributes.

Microservice-based model-driven architecture for implementing modeling and model-transformation as a service

Adel Vahdati, Raman Ramsin

Department of Computer Engineering, Sharif University of Technology, Azadi Avenue, Tehran, Iran
adel.vahdati97@sharif.edu, ramsin@sharif.edu

Abstract

Model-driven engineering (MDE) is a popular software engineering approach that emphasizes the use of models throughout the software development lifecycle. The goal of MDE is to reduce accidental complexity, and thereby leads to several benefits, including automatic code generation; however, there are several challenges that make it difficult to use in industry. These challenges include scalability issues, such as model size and complexity, and issues related to collaborative modeling and artifact reuse. In addition, the need remains for a clear-cut framework and methodology for applying MDE in an agile context. We propose a model-driven architecture that utilizes model-driven development (MDD) practices and service-oriented approaches through the microservice architecture. To this aim, we introduce the idea of multi-level modeling as a service and model transformation as a service and develop a model-driven platform (called MDDPlatform) to address issues related to reusability, scalability, and collaboration in software development. We have evaluated the proposed approach through a comparative analysis based on relevant metrics and criteria, and also through a case study that assessed the impact of the approach on relevant quality attributes.

Keywords: Model-Driven Engineering, Microservice, Model Transformation, Domain-Specific Language

1. Introduction

Model-driven engineering (MDE) is a popular approach in software engineering that utilizes models as primary components for analysis, design, and implementation. The system's characteristics are described using abstract models at various levels of abstraction, which are then transformed into lower-level abstract models closer to the solution space. Finally, executable code is generated through model-to-text transformations (Alam et al., 2018; Rodrigues da Silva, 2015).

Modern software systems require collaboration between stakeholders, domain experts, and development teams. Domain experts use high-level abstract concepts to express requirements, while development teams use low-level concrete concepts to implement solutions. Filling the gap between these levels of abstraction introduces various challenges when requirements change (Alam et al., 2018; Combemale et al., 2014; Rodrigues da Silva, 2015). MDE was introduced to reduce accidental complexity in software systems (Alam et al., 2018), mainly through the use of models and model transformations throughout the software development lifecycle to bridge the gap between high-level requirements and low-level implementation concepts (Kusel et al., 2015; Rodrigues da Silva, 2015; Wagelaar et al., 2011).

As software systems become larger and more complex, the constraints and challenges of MDE are intensified (Di Rocco et al., 2016; D. S. Kolovos et al., 2015). Challenges such as lack of scalability when working with medium-sized models, as well as weaknesses in supporting collaboration and interaction throughout the MDE process, have proven especially troublesome (Bucchiarone et al., 2020). Large and complex systems have various dimensions that require different languages to accurately describe, probably created using different technologies. Therefore, combining these languages and presenting a simplified view of models and a hierarchical structure of their elements can be challenging (D. Kolovos et al., 2013).

One of the challenges in this field is the limited support provided for reusing previously produced artifacts during the MDE process. In addition to storage repositories, advanced queries of artifacts based on their relevant metamodels, domains and creation phases are necessary to facilitate the reuse of existing modeling artifacts (Alam et al., 2018). Managing public and private repositories and integrating modeling artifacts with different access policies is another issue that remains unresolved. Storing, retrieving, and displaying large models effectively are other issues of interest in the field (Basciani et al., 2016; D. Kolovos et al., 2013) Various languages and tools have been proposed for transforming models; however, the tight coupling between the transformations and the specific metamodels of input and output models poses serious challenges in adapting existing transformations to new domains and metamodels, making the reuse of model transformations difficult (Cuadrado et al., 2014; Kusel et al., 2015).

The aim of this research was to enhance MDE by utilizing a service-oriented approach, with a focus on scalability, reusability, and collaboration. The proposed model-driven architecture is based on the microservice architecture and supports the notions of "multilevel modeling as a service" and "model transformation as a service". We have developed a platform, called MDDPlatform, to facilitate MDD and realize the proposed approach. The proposed approach and MDDPlatform were then evaluated through comparative analysis and a case study. The results of these evaluations have shown that reusability, scalability, and collaboration are indeed enhanced through the use of our proposed approach.

The paper is structured as follow. In Section 2, we review existing research on model-driven services. In Section 3, the challenges of MDE as to reusability, scalability, and collaboration are discussed. In Section 4, the proposed model-driven architecture is described, and its support for multilevel modeling and model transformation as a service are elaborated upon. MDDPlatform is introduced in Section 5, and Section 6 examines the relevant criteria for evaluating MDE solutions. In Section 7, we analyze and compare MDDPlatform to other model-driven services based on the relevant evaluation metrics. In Section 8, the case study conducted for evaluating MDDPlatform is described and its results are presented and analyzed. In Section 9, the threats to the validity of this research have been examined and addressed, and in Section 10, conclusions are presented and the future work is outlined.

2. Related Works

The introduction of service-oriented architecture has resolved issues with monolithic systems, but as software development progressed, microservices were introduced. Microservices aim to create distributed systems consisting of a set of self-contained services that provide capabilities and functionalities beyond individual services through interaction and integration. These services have well-defined interfaces, thus increasing system flexibility and adaptability, and facilitating service replacement and deployment. Microservices can be deployed independently and have their own resources and databases. Team productivity increases because the system architecture aligns with the structure of development teams, as each service is maintained by one team. The independence of microservices increases quality and safety by allowing services to be examined independently and scaled according to demand at runtime (Dragoni et al., 2017; Rademacher et al., 2017).

Service-oriented architecture and microservices have different objectives. Service-oriented architecture aims to integrate software assets at a higher level to achieve business process goals, regardless of the internal structure of the software programs. On the other hand, microservices aim to enhance software development, delivery, and deployment by creating self-contained services with well-defined interfaces that interact with each other. The internal structure of the software is crucial for microservices, and each service is a manageable and deployable unit (Heinrich et al., 2017). We will examine the main research efforts on service-oriented MDE in this section.

The MDEForge platform was developed to offer a community-based modeling repository with advanced searching mechanisms and the capability to use model management tools as a service. It was designed to be scalable and extensible, allowing users to upload, download, delete, and search for metamodels, models, and model transformations. The platform also supports the execution of transformation chains. However, it does not support artifact versioning. The main focus of the MDEForge platform is on providing a repository of artifacts, but it does not allow for the creation of new artifacts. Users must upload appropriate editors based on the relevant metamodels if they wish to edit models. For instance, if the user wants to modify a model, they need to upload the JAR file that corresponds to the desired editor (Basciani et al., 2014; Popoola et al., 2017).

The creation of services in model-driven service-oriented architectures is mainly based on models. These models contain valuable information that can be used by the services responsible for managing, adapting, and monitoring services at runtime. However, if the corresponding metamodels of these services are modified, the services need to be manually rebuilt and redeployed. To address this issue, the concept of model-aware services and repositories was introduced in MORSE (Holmes et al., 2012), a service-oriented environment. The primary objective of MORSE is to achieve automatic adaptability of services at runtime based on model changes and creating services from models, while hiding versioning complexities from the users (Holmes et al., 2012; Popoola et al., 2017).

The AToMPM (Corley & Syriani, 2014; Eugene Syriani et al., n.d.) modeling platform allows multiple users to collaborate and share modeling artifacts, with each user having their own view of the

model. It can be used on the cloud without requiring software installation and supports model transformation. However, it does not support versioning or the merging of models (Popoola et al., 2017).

The web-based platform WebGME (Maróti et al., 2014) aims to aid collaboration and cooperation in designing domain-specific modeling tools and can be used on the cloud. It allows for versioning, inheritance and merging of models by creating a copy of the model and establishing a parent-child relationship between the original and copy versions. WebGME also supports model transformation but does not support code generation or the merging of models (Popoola et al., 2017).

The Eclipse project EMF.Cloud (*EMF.Cloud*, n.d.) was created to bring EMF capabilities to web platforms, allowing the use of the EMF modeling framework in web applications. The objective of EMF.Cloud is to offer a universal and application-agnostic set of tools for presenting, modifying, handling, transmitting, and managing EMF-based models on the web. To achieve this, they have developed a web-based modeling tool that utilizes Ecore and a "model server", which can handle multiple clients and distribute the model's state. The server has a command-driven interface and supports socket notification.

3. MDE Challenges

In this section, we will examine the challenges of MDD, with an emphasis on reusability, scalability, and collaboration.

3.1. Reusability

There are several methods for reusing models, including copying them, importing them as proxies, using them as black boxes with communication through an interface, and exposing all elements and relationships for white-box reuse. When reusing models, it is important to consider their interdependencies to ensure compatibility (Alam et al., 2018).

In order to create reusable model transformations, it is necessary to have abstraction mechanisms that can work with different metamodels and apply changes to ensure compatibility with a specific metamodel. Composing model transformations and creating new ones is also important (Wimmer et al., 2010). However, the challenge lies in the structural differences between metamodels, which describe the same concepts differently. This makes it difficult to use existing transformations for other metamodels without manual adaptation. To increase reuse and avoid manual work, an automatic solution is needed to manage heterogeneity between metamodels (Wimmer et al., 2012).

According to Kusel and colleagues (Cuadrado et al., 2014; Kusel et al., 2015), there are four main challenges that hinder the reuse of model transformations. One of these challenges is that model transformations are closely linked to specific metamodels, making it difficult to use them for metamodels that share similar meanings but have different structures. Additionally, the lack of meta-information and repositories for locating model transformations poses another obstacle. Adapting existing transformations to new contexts is also problematic due to various limitations and difficulties, and there is inadequate support for integrating transformations.

Reusability of model transformations can be achieved through various techniques and strategies. One approach involves adjusting the initial metamodels that the transformation conforms to, according to the new metamodel that the transformation is reused for (Bruel et al., 2020). This can be accomplished through model typing (Steel & Jézéquel, 2007), which defines subtyping relationships between metamodels. Another strategy is to use posteriori typing (Lara & Guerra, 2017), which involves reclassifying the objects based on classes other than their creator class. The Concepts approach (Bruel et al., 2020) involves rewriting the model transformation using a higher-order transformation (HOT) so that it can be reused for a specific metamodel. In this approach, a Concept is similar to a metamodel but with the difference that the types of its elements are parameterized and must be bound to the elements of a metamodel (Bruel et al., 2020).

Model transformation patterns and operators create model transformation code based on design patterns and a combination of mapping operators. A general set of mapping operators for adapting one metamodel to another is provided in the form of a library (Legros et al., 2009). The multi-level modeling approach defines different levels of abstraction, allowing for greater flexibility in modeling. In this approach, model management operations related to higher levels are defined in a general way and then used directly or indirectly for existing instances at lower levels (Bruel et al., 2020). Lastly, constraints and required types are extracted from model transformations and expressed in the form of a model, which acts as a requirements model. If a metamodel satisfies the constraints and supports the types present in this model, then the model transformation can be applied to models conforming to that metamodel (Bruel et al., 2020).

3.2. Scalability

As MDE is applied to larger and more complex systems, there is a growing need for a suitable platform to efficiently develop, manage, share, and store large models (D. Kolovos et al., 2013; D. S. Kolovos et al., 2015). This requires flexible and scalable solutions and techniques that can accommodate the increased computational resources, memory, and storage space required for such systems (Di Rocco et al., 2016). Scalability in model-driven software engineering encompasses various dimensions, including team size, number and size of artifacts, and the complexity and size of modeling languages and tasks (D. Kolovos et al., 2013). To achieve scalability, we need to create large models and domain-specific languages systematically, enable collaboration among large teams, provide query and transformation tools that can handle large models, and have an infrastructure that can efficiently store, index, and retrieve model elements (D. Kolovos et al., 2013; D. S. Kolovos et al., 2015).

The current state of models and modeling languages makes them difficult to scale, and combining models described by different languages requires modularization techniques and greater flexibility in language integration. Challenges related to scalability include breaking large models into parts, maintaining consistency between them, and validating them. To address these challenges, models can be represented as a hierarchical structure and organized at different abstraction levels. Different languages may be needed to express the various aspects of a complex system, requiring

modeling environments that can reuse parts of the metamodels created with heterogeneous technologies (D. Kolovos et al., 2013).

The commonly used XMI format for storing models in the EMF framework has improved interoperability between modeling tools, but it has its own limitations. Due to its XML-based nature, XMI requires loading the entire file into memory, making it unsuitable for effective display, partial loading, or lazy loading of models. Furthermore, XMI is verbose and results in larger file sizes. To address these issues, an optimized model storage solution is needed that allows for access to portions of a model and its elements, as well as lazy loading. Storing entire models in a single file is not efficient for network resource usage and conflict management when making changes to artifacts stored in a central repository. This approach requires transferring the entire file between local and remote machines for every change, and can create multiple conflicts if optimistic locking mechanisms are used (D. Kolovos et al., 2013; D. S. Kolovos et al., 2015).

Various methods have been proposed to improve the efficiency of model transformations and resource usage. One such method is gradual and incremental transformation, which involves using tools to capture changes in models and keep the source and target models synchronized. Loading and transforming models should be tailored to the specific needs of the model, as large models may not fit in memory in their entirety. Lazy loading and distributed computation can aid in managing these models. On-demand generation of model elements through a network of model transformations and reactive transformation engines can help address scalability challenges by performing the necessary computations for transformations and maintaining compatibility between models (D. Kolovos et al., 2013; D. S. Kolovos et al., 2015).

3.3. Collaboration

The collaborative model-driven software engineering approach is based on using models as the primary artifact to drive software development and other model-driven tasks in the software engineering process. This approach is built on three main pillars: model management, collaboration, and communication. To implement this approach effectively, a proper infrastructure is needed to manage models throughout the software development lifecycle, as well as tools to facilitate stakeholder participation and collaboration in model generation and manipulation, and mechanisms for modelers to communicate and stay informed about each other's activities. Managing models requires a repository for storing the models and related metadata, as well as modeling tools to create, update, delete, and share models in various formats (Franzago et al., 2018).

To enable participation and collaboration, we also need a multi-user environment that enables online/offline collaboration, and a version control system to manage conflicts and different versions of models. Conflict management is one of the main challenges in this area. Another challenge is the synchronization and dissemination of changes made by other members during collaborative modeling. Model repositories currently offer limited support for branching, model comparison, and conflict management. To address these challenges, conflict resolution should be

automated and should take into account model compatibility at the domain level. Additionally, model changes should adhere to conflict-free policies (Franzago et al., 2018; D. Kolovos et al., 2013).

In collaborative environments, it is important to identify who is working on which part of the models and who the previous contributors are. To achieve this, interaction should be definable and traceable to the models. Indirect communication can be achieved through documents, comments, and resources such as a shared knowledge base. Documenting basic concerns gives new participants the opportunity to overcome the initial obstacles to reuse. Comments and explanations play an important role in domain-specific modeling languages, but their connection to different elements of the models should be maintained and managed separately to prevent disruption in the modeling process (Alam et al., 2018).

To improve the collaboration and participation of members of large teams in the modeling process, a clear framework and methodology is required that facilitates the interaction of different stakeholders (D. Kolovos et al., 2013). Agile methodologies promote close collaboration and interaction between stakeholders and development teams. Currently, agile principles/values have been increasingly adopted and utilized by the industry. Therefore, companies only move towards MDE if it can create value in the agile era. To improve MDE solutions and cover the social aspects of collaborative modeling in heterogeneous multi-disciplinary environments, we need to focus on the social aspects of the collaborative modeling process (Bucchiarone et al., 2020).

4. Proposed Solution

The following section covers the topic of model-driven architectures, starting with the MDA, proposed by OMG, and our own proposed architecture, the Model-Driven Onion Architecture. We also explore current modeling approaches and show how our architecture can overcome some of the limitations of traditional methods by introducing a new dimension of relationships between the elements in different models. Lastly, we discuss the concepts of multi-level modeling as a service and model transformation as a service, which are essential components of the Model-Driven Onion Architecture.

4.1. Proposed Model-Driven Onion Architecture

Development of software using a model-driven approach requires a well-designed architecture. MDA was introduced by OMG to address this need, emphasizing the creation of models at different levels of abstraction and transforming them into more detailed models, ultimately resulting in a solution model or code. MDA supports three levels of abstraction: computation-independent model (CIM), platform-independent model (PIM), and platform-specific model (PSM) (Rodrigues da Silva, 2015).

MDA's approach to software development involves transforming models from higher levels of abstraction to lower ones through a chain of model-to-model transformations. The process starts with CIM models, which are transformed into PIM models, and then into PSM models. Finally, code is generated using model-to-text transformations. However, this approach presents challenges

such as the need to synchronize higher-level models after changes are made to lower-level ones, requiring round-trip engineering (Diskin et al., 2014). In practice, MDD in the proposed OMG architecture involves three main phases: model construction, model transformation, and round-trip engineering; round-trip engineering is still the main challenge in this approach (Asadi & Ramsin, 2008; Diskin et al., 2014; Hailpern & Tarr, 2006; Shin, 2019). In fact, we are witnessing a combination of top-down and bottom-up transformations on models at different abstraction levels with strong interdependencies. Another challenge is that models created at different levels of abstraction often overlap significantly (Diskin et al., 2014).

To tackle the challenges of round-trip engineering and minimize model overlap, we introduce the Model-Driven Onion Architecture. The architecture consists of CIM models at its core, which describe the problem domain without being dependent on the design/solution space. The PIM models in the second layer only focus on design concerns and avoid redefining attributes and features of a concept already described at the CIM level. To achieve this, a new dimension of relationship is proposed, allowing access to these specifications while maintaining loose coupling from upper layer models (lower level of abstraction) to lower layer ones (higher level of abstraction). PSM models are introduced in the third layer, where only implementation concerns are modeled, and a similar approach is used to accessing information from the PIM layer. The Code is at the fourth layer, where the solution is described as a model containing the relevant aspects of the code.

Figure 1 illustrates both the Model-Driven Onion Architecture and the MDA architecture proposed by OMG. The OMG architecture follows a top-to-bottom dependency direction, starting from the CIM layer down to the PIM layer, and then to the PSM layer. In contrast, the Model-Driven Onion Architecture has an inward dependency direction towards the core, with the PIM layer depending on the CIM layer, the PSM layer depending on the PIM layer, and the Code layer depending on the PSM layer.

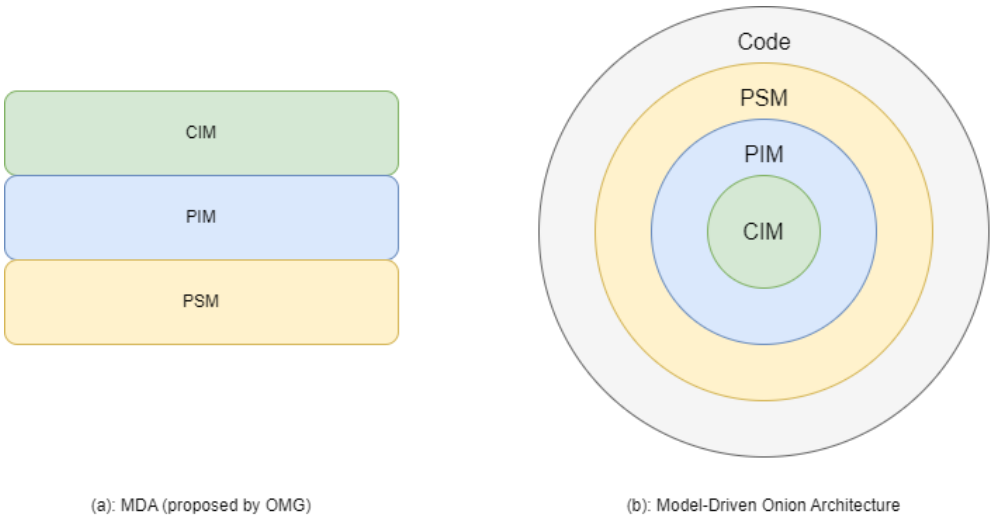

Figure 1: Model-driven Onion Architecture compared to MDA

4.2. A new dimension of relationships

This section introduces a new dimension of relationships between the elements of two models. It is important to first understand the different modeling approaches currently in use.

The conventional modeling method is structured in two levels, where the DSL's abstract syntax is defined in a metamodel, and the problem model is explained by generating instances of the types defined in the metamodel. This architecture is supported by the EMF modeling framework. The metamodel is established at the meta-level using features like data types, type definition, type inheritance, property definition, and relationship cardinality. Models are then produced by instantiating the metamodel elements. It's crucial to note that these features are only accessible at the metamodel level and not at the model level. As a result, if a new type is required, it must be explicitly defined in the metamodel (Lara et al., 2014).

The Orthogonal Classification Architecture (OCA) is a technique proposed to extend the standard approach of two-level modeling. OCA proposes two orthogonal type systems for modeling elements, one based on ontology and the other based on linguistics. In ontology-based classification, instantiations are based on the logical dimension of the elements defined in the hierarchy of existence. In linguistics-based typing, the physical dimension of the element is considered and refers to the concepts required to construct and represent the element (Lara et al., 2014).

OCA can be explained by an example in the object-oriented paradigm, where the "Is A" relationship between a concept and its instances can serve as the basis for classification. In other words, the instances of a concept have characteristics and properties that are described in the specification of that concept. This aspect of classification stems from an ontological view of concepts and domain entities. The relationship between "Book" as a concept and "Les Miserables" can be classified from this perspective. In the modeling context, these concepts and entities must be represented in some way. In object-oriented modeling, concepts and types are modeled as classes, and instances of a concept or type are represented as objects. Therefore, we are also faced with the second type of classification here that stems from the way concepts and entities in the modeling domain are represented.

In addition to this, we propose a third dimension of classification called the "Relational Dimension". This dimension is based on the unidirectional relationship between a domain object and a concept that is not considered an instance in either an ontological or linguistic dimension but can be understood at a higher level. The "Relational Dimension" is defined by two properties: the "Relation Name" and the "Relation Target", which express the name and meaning of the relationship and the associated concept, respectively.

The "Relational Dimension" in the Model-Driven Onion Architecture allows us to connect domain objects at the PIM level with domain concepts at the CIM level without compromising the independence of the latter from design concerns of the solution space. By combining domain objects in the design-specific model with information about domain concepts at the CIM level

through model transformations at the PIM level, we can create a richer and refined model of domain concepts at the PIM level. The choice of which domain concepts to use at the CIM level is determined by the information described in the "Relational Dimension". If the design-specific model is modified for some reason, these changes are not propagated to the CIM-level models, and the refined model created at the PIM level is updated by re-executing the transformations. Similarly, at the PSM level, only implementation concerns are described in a model of domain objects, and the connection between these objects and higher-level concepts in the PIM model is established using the "Relational Dimension". Here too, changes in implementation concerns at the PSM level are not propagated to PIM models.

The "Relational Dimension" in the proposed architecture has the potential to solve the issue of round-trip engineering and prevent changes from propagating from lower-level abstraction models to higher-level ones. It also reduces overlapping of models created at different levels of abstraction. While it may not completely resolve the problem of model overlap, it can reduce rework and eliminate duplication if the metamodels at the PIM and PSM levels are designed with the Single Responsibility Principle (SRP) and separation of concerns in mind. If metamodels are designed based on these best practices, the "Relational Dimension" has the potential to avoid redundancy and waste.

4.3. Multi-level modeling as a service and model transformation as a service

Up to this point, we have demonstrated how the Relational Dimension can establish a loosely coupled relationship between the domain objects in a model at a lower level of abstraction and the domain concepts in a model at a higher level of abstraction. However, the challenge lies in accessing the features of the referenced domain concept in the Relational Dimension when it is described at a higher level of abstraction in a separate model. To address this problem, we propose the concept of multi-level modeling as a service. In the Model-Driven Onion Architecture, each layer provides services to the layer above it.

By providing the necessary functionalities as services, each layer can act as a facade and provide access for its upper layer to retrieve the specification of models described at that level of abstraction. This allows model transformations at the PIM level to retrieve the desired features and properties of domain concepts at the CIM level using the information stored in the Relational Dimension component of domain objects in PIM models, thus generating a refined model at the PIM level. Similarly, the PIM layer acts as an interface on models at that level and provides access to the model specification for its upper layer (PSM layer). However, each layer can only describe and update model specifications through the components and model transformations residing in that layer, and higher layers cannot change model specifications at lower layers.

To develop software using a model-driven approach, we need to create and execute a series of model transformations at different levels of abstraction. To achieve this, a high-level process is defined to guide the sequence of transformation execution and model refinement. This high-level process provides the necessary functionalities for executing transformations in the form of

services. It can refine PIM-level models by calling and orchestrating CIM-level model services and PIM-level transformation services, and merge design-specific models at the PIM-level with required information from CIM-level models. Subsequently, it can refine PSM-level models based on implementation-specific models at the PSM-level and required information from PIM-level models. Finally, it can generate executable code using model-to-text transformations. Table 1 categorizes the model transformation services and their input and output models.

Table 1: Abstraction levels of transformations and input/output models in the proposed model-driven onion architecture

Abstraction level of transformation	Abstraction level of input model	Abstraction level of output model
CIM-level Model Transformation	CIM	CIM
PIM-level Model Transformation	CIM	PIM
	PIM	PIM
	CIM, PIM	PIM
PSM-level Model Transformation	PIM	PSM
	PSM	PSM
	PIM, PSM	PSM
Code-level Model Transformation	PSM	Code
	Code	Code
	PSM, Code	Code

5. Proposed MDDPlatform

In this section, we present a model-driven platform based on the microservice architecture, which we have chosen to call MDDPlatform. We have developed MDDPlatform as a proof of concept. This platform is currently composed of six primary microservices that are utilized to manage the problem domain, sub-domains, domain-specific modeling languages, domain models, MDD processes and model transformation patterns, and also visualization of models in graphical diagrams. Furthermore, the services offered by this platform are accessible to users through an API Client user interface, which can be accessed via a browser. The high-level structure of the MDDPlatform architecture is depicted in Figure 2. In the following sections, we will introduce each of these microservices and their respective tasks. Further details have been provided in an Appendices file that has been made available on Mendeley Data (Vahdati & Ramsin, 2023).

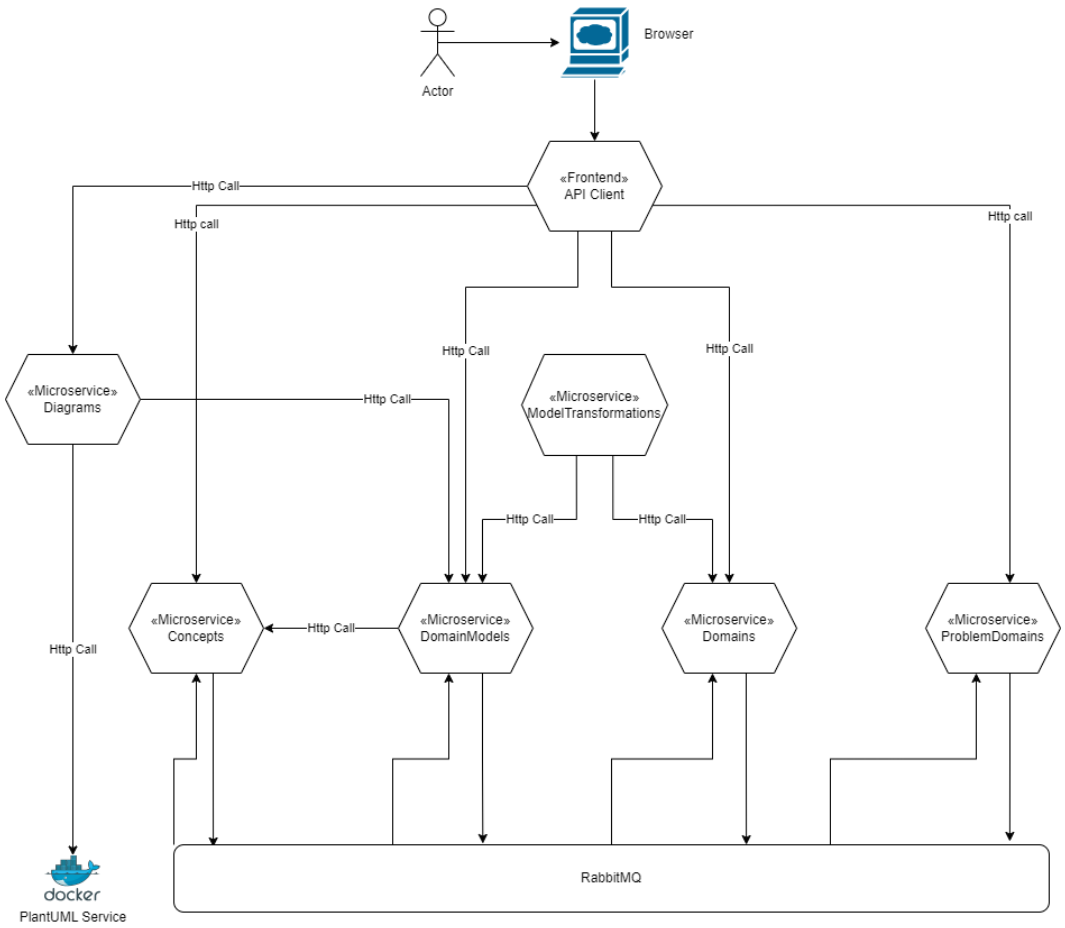

Figure 2: MDDPlatform's high-level microservice-based architecture

5.1. Concepts microservice

In order to describe the problem and solution domain, a modeling language is necessary. To create a modeling language, an abstract syntax must first be established by defining a metamodel. The Concepts microservice allows for the creation of domain-specific languages and their constituent concepts. This microservice defines the attributes and relationships of these concepts and uses them to define the domain-specific language. The Concepts microservice also enables sharing of concepts and languages, as well as the creation of new languages based on previously described concepts. Additionally, it is possible to search for domain-specific languages based on their constituent concepts. The high-level structure of the Concepts microservice is shown in Figure 3.

Using this microservice, 13 metamodels have been created at different levels of abstraction to suit various modeling goals. At the CIM level, two metamodels have been proposed: the CRAC metamodel for exploring and describing the problem domain using the CRAC method (Vahdati & Ramsin, 2022), and the CIM metamodel for describing domain concepts, their properties, and relationships. At the PIM level, several metamodels have been provided for design purposes, including the core domain metamodel, domain service metamodel, repository pattern metamodel, CQRS pattern metamodel, API controller pattern metamodel, minimal domain-driven design

metamodel, and minimal database metamodel, each describing a different aspect of the solution domain. At the PSM level, two metamodels have been provided for implementing solutions and generating code in the C# language in .NET framework. At the Code level, a simple metamodel has been established for modeling project structure, folders, files, and their content to generate solution code in a ZIP file format.

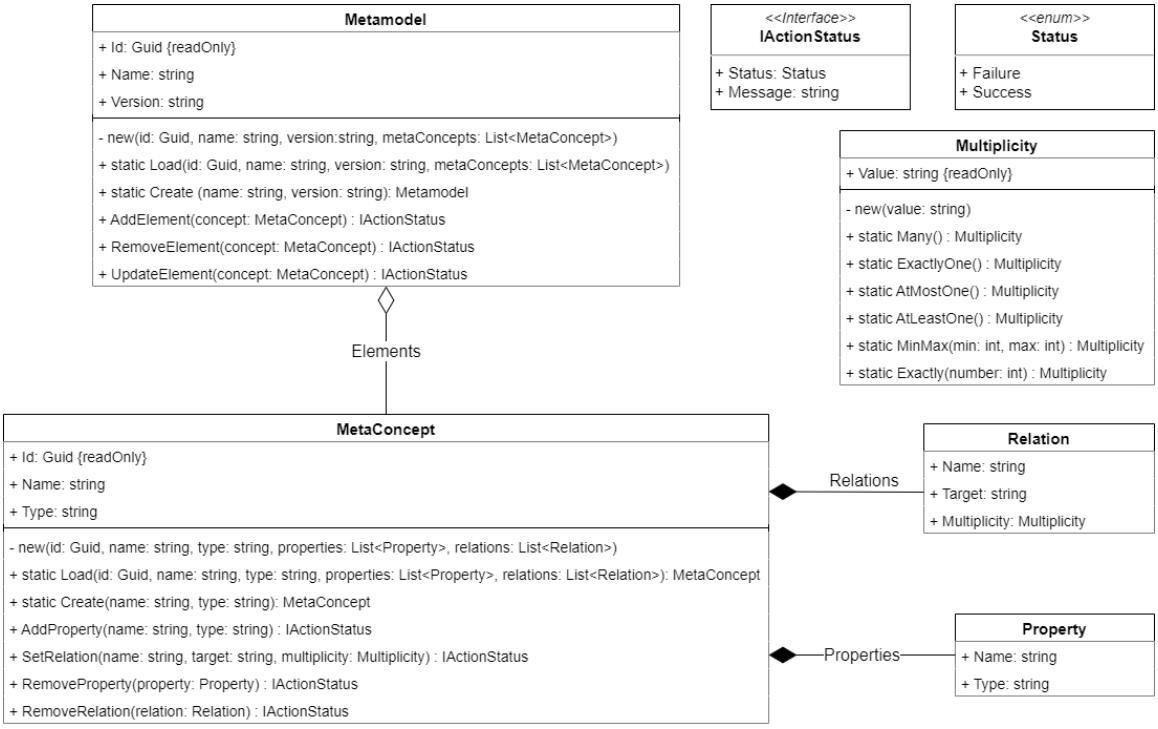

Figure 3: Structure of the main concepts in the Concepts microservice

5.2. ProblemDomains microservice

The ProblemDomains microservice is responsible for managing the problem domain at the highest level. This microservice enables us to decompose the problem domain into its distinct subdomains after it has been defined. Additionally, it manages team members and assigns them to subdomains, as well as determining their roles and access levels. Furthermore, this microservice is responsible for managing the policy of sharing and access to artifacts at the highest level.

5.3. Domains microservice

The Domains microservice is responsible for managing subdomains. Each subdomain can be described from a different perspective using various domain-specific modeling languages. This microservice provides functions such as creating models at different levels of abstraction, deleting a model, and listing the models that describe a subdomain. Moreover, access permissions to some operations, such as creating, updating, deleting, and viewing a model specification, are managed by this microservice.

5.4. DomainModels microservice

The DomainModels microservice is responsible for facilitating the creation of domain models using specific modeling languages. This includes creating instances of concepts, establishing relationships between objects, and assigning property values in accordance with the metamodel and its constraints. It also provides functions for manipulating and updating models, as well as retrieving all or part of the model specification. The microservice offers querying services for elements with specific types or relationships, as well as coarse-grained operations for model manipulation. The ModelTransformations microservice uses these functions to generate or refine output models based on transformation patterns and input model information. The Diagrams microservice also utilizes querying and information retrieval services to display model descriptions visually. Figure 4 illustrates the main concepts of the DomainModels microservice.

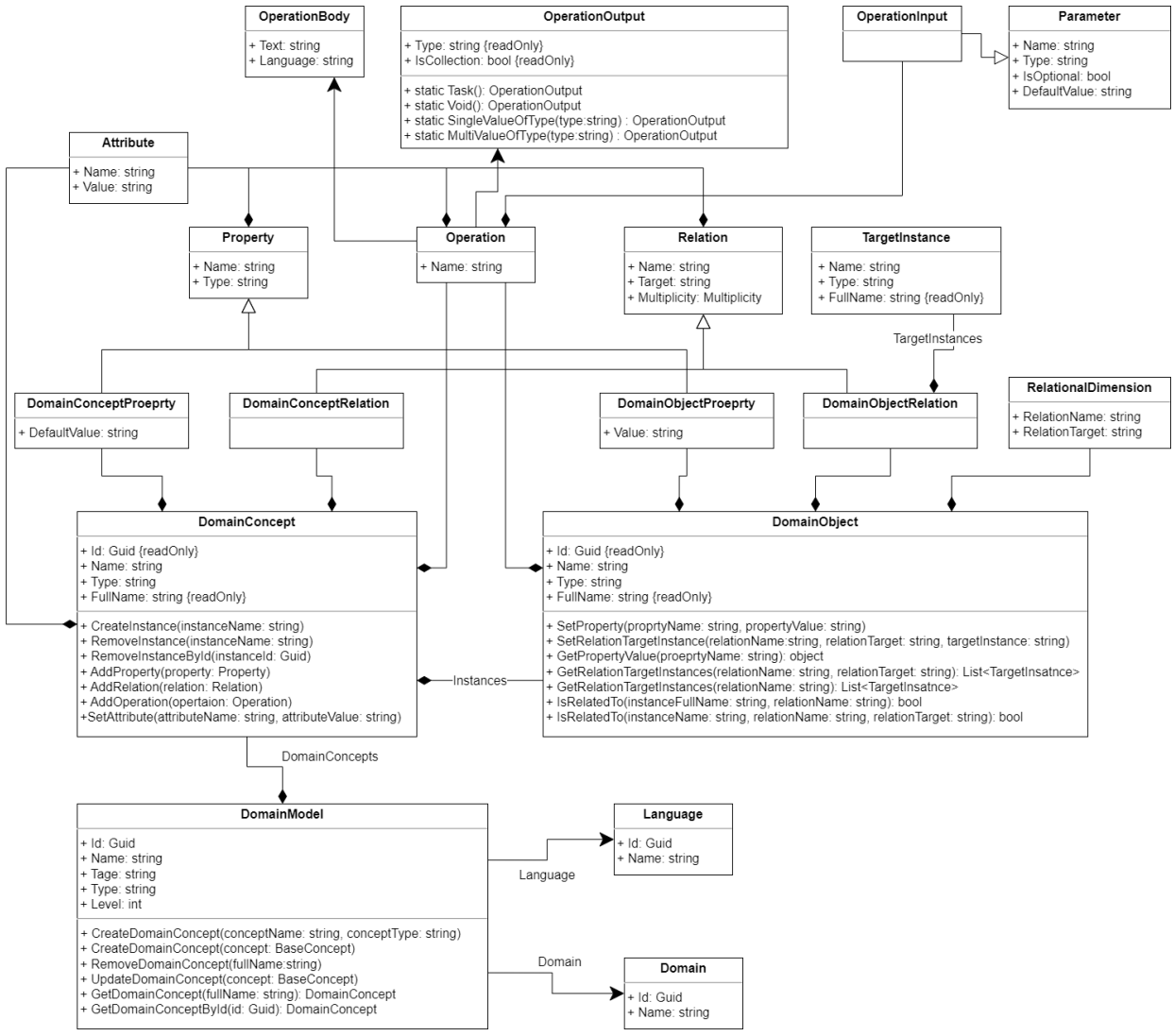

Figure 4: Structure of the main concepts in the DomainModels microservice

The data structure and storage technology of models can vary depending on the modeling goals and performance requirements, and there is no one-size-fits-all solution. For small and medium-sized projects, NoSQL databases like MongoDB can efficiently store, retrieve, and manipulate model specification in memory. However, for larger models, it may be necessary to use graph databases or break the model into parts. To address this issue, multiple DomainModels microservices are used, each offering an optimal solution for storing and managing model information using separate storage technology. The API Gateway architectural pattern (Richardson, 2018) is used to hide this complexity from the client. For example, the ModelTransformations microservice are not concerned with the storage technology of the input and output model, the data structure of the model specification, and how they are stored or retrieved.

The API Gateway pattern serves as the entry point for APIs and directs incoming requests to internal microservices based on specific criteria. This means that external clients and other microservices are only able to communicate with internal microservices through the API Gateway, which hides the details of microservices and request distribution. To ensure that the model-driven platform is extensible, two microservice architectural patterns - self-registration and server-side discovery - can be used together for service discovery and interaction. With the self-registration pattern, new versions of the DomainModels microservice can register themselves in the service registry through the API Gateway. The server-side discovery pattern allows clients to send requests to the API Gateway, which then finds the appropriate version of the DomainModels microservice to handle the request and returns the response to the client. The overall architecture is shown in Figure 5.

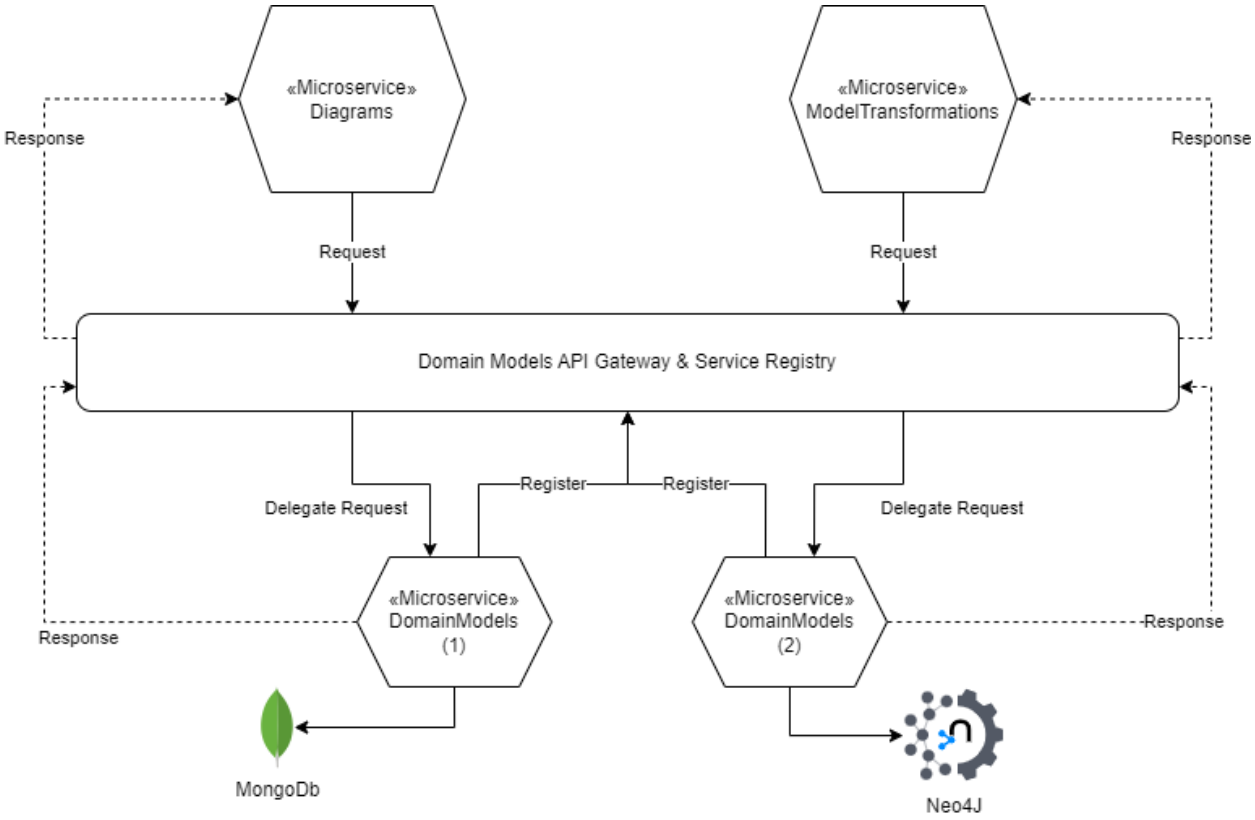

Figure 5: Managing different versions of the DomainModels microservice

5.5. ModelTransformations Microservice

Our proposal is to create a service that provides transformation patterns and functionalities for model transformations, with the goal of improving their reusability. In MDD, models are created at different levels of abstraction and transformed into more detailed models using model transformations. These transformations can be abstracted into patterns, which can be reused across different domains. To define a transformation pattern, we need to identify the required parameters, such as the types of entities and their relationships. By parameterizing the pattern, we can create instances of it for executing transformations. To implement this idea, we have developed a microservice called ModelTransformations, which includes five components: transformation pattern, pattern instance, pattern instance template, transformation pipeline, and transformation process.

The transformation patterns are declarative in nature, where the user specifies the desired outcome without specifying how it should be done. These patterns are provided as services and documented with pattern title, category, problem statement, required parameters, solution, and an example of executing the transformation on an input model. The transformation patterns fall into four categories: 1) converting a model of domain objects into another model of domain objects, 2) transforming a model of domain objects into a model of domain concepts, 3) merging a model of domain objects with a model of domain concepts based on “Relational Dimension” specification, and transforming model specification into text.

By creating an instance of a transformation pattern and assigning parameter values based on the problem domain, the user can transform one or more input models into an output model. The patterns are designed as deployable web services, with the pattern parameters defining the message structure that needs to be sent via an HTTP request to receive the model transformation service. The user only needs to select the desired pattern and create an instance of it, and the actual execution of the transformation is handled by the ModelTransformations microservice.

If a user creates instances of transformation patterns according to specific solution architecture and design patterns, they may wish to reuse those patterns for another project with similar architecture and modeling language. To reuse pattern instances in other projects, a pattern instance template is introduced. This template includes all parameter values except for those related to input and output models, which are configured at runtime based on the specific project. Figure 6 illustrates the structure of the transformation pattern in the ModelTransformations microservice.

The service also includes a model transformation pipeline component that executes a chain of transformations defined at different levels of abstraction. Each pipeline has one or more stages, which can be automated or require human intervention. If a stage is automated, the corresponding pattern instance is automatically executed, and if successful, the next stage is called. However, if a stage requires human intervention, the execution of the pipeline is paused until the user performs the tasks related to that stage and requests to continue the pipeline's execution. The structure of the pipeline in the ModelTransformations microservice is shown in Figure 7.

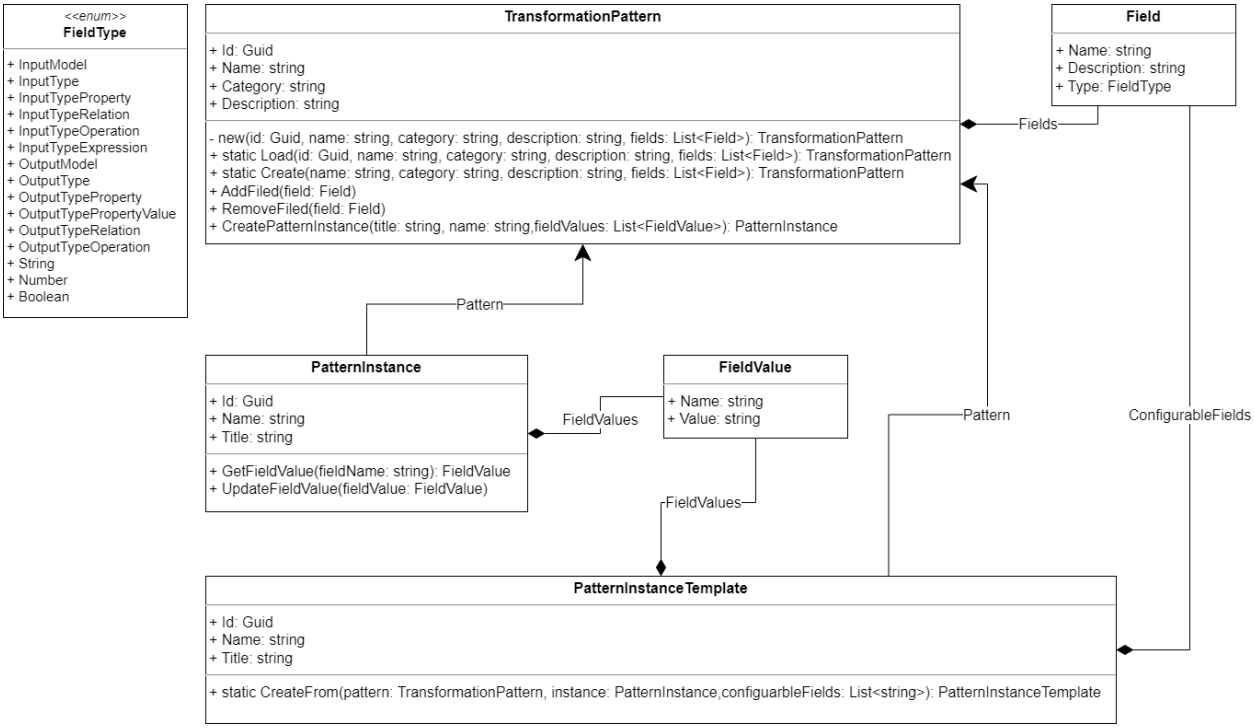

Figure 6: Transformation pattern structure and related concepts

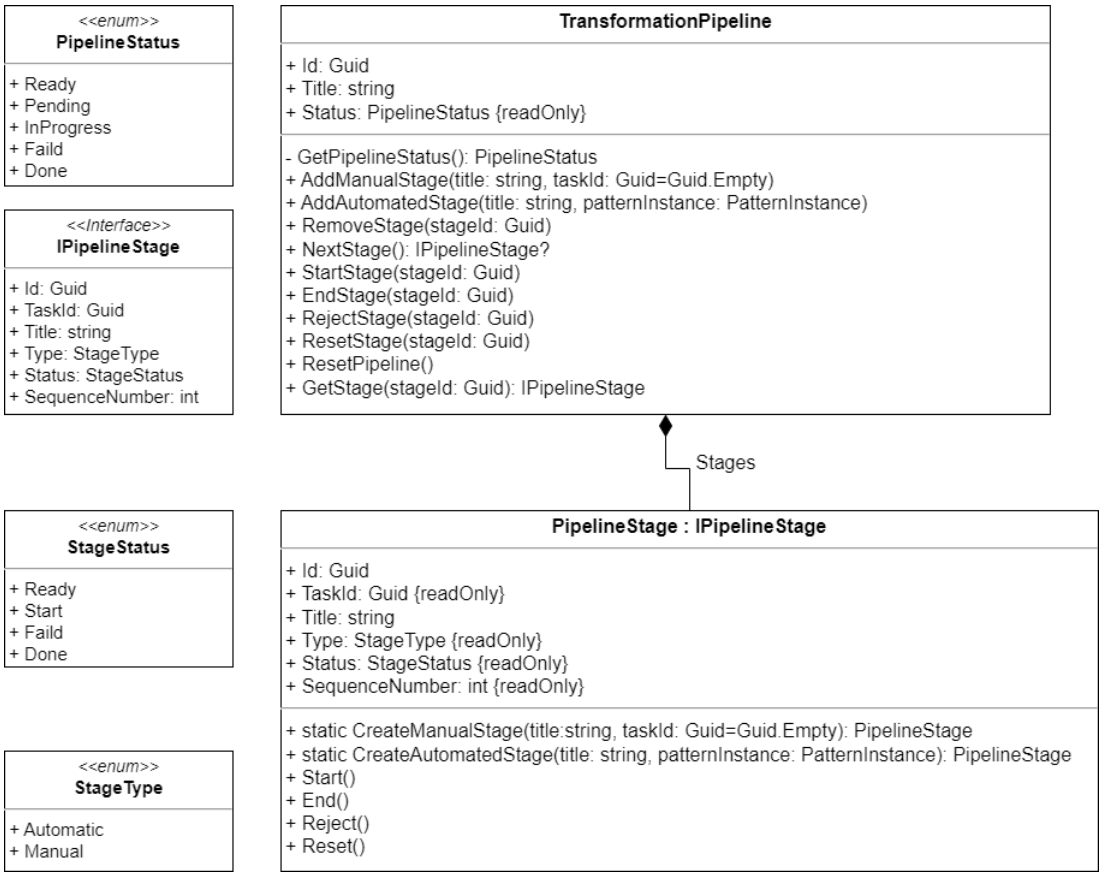

Figure 7: Structure of the pipeline in ModelTransformations microservice

The creation of software using a model-driven approach requires a specific process to guide the creation and refinement of models and the order of executing model transformations. The ModelTransformations microservice allows for defining, configuring, and executing high-level processes in different projects, consisting of phases, activities, and tasks. These tasks can be automated or manual, involving model transformations or adding analysis and design information to refine the models. Activities can be defined using transformation pipelines or composed of automated and manual tasks in a specific order. The high-level structure of the process and related concepts are illustrated in Figure 8.

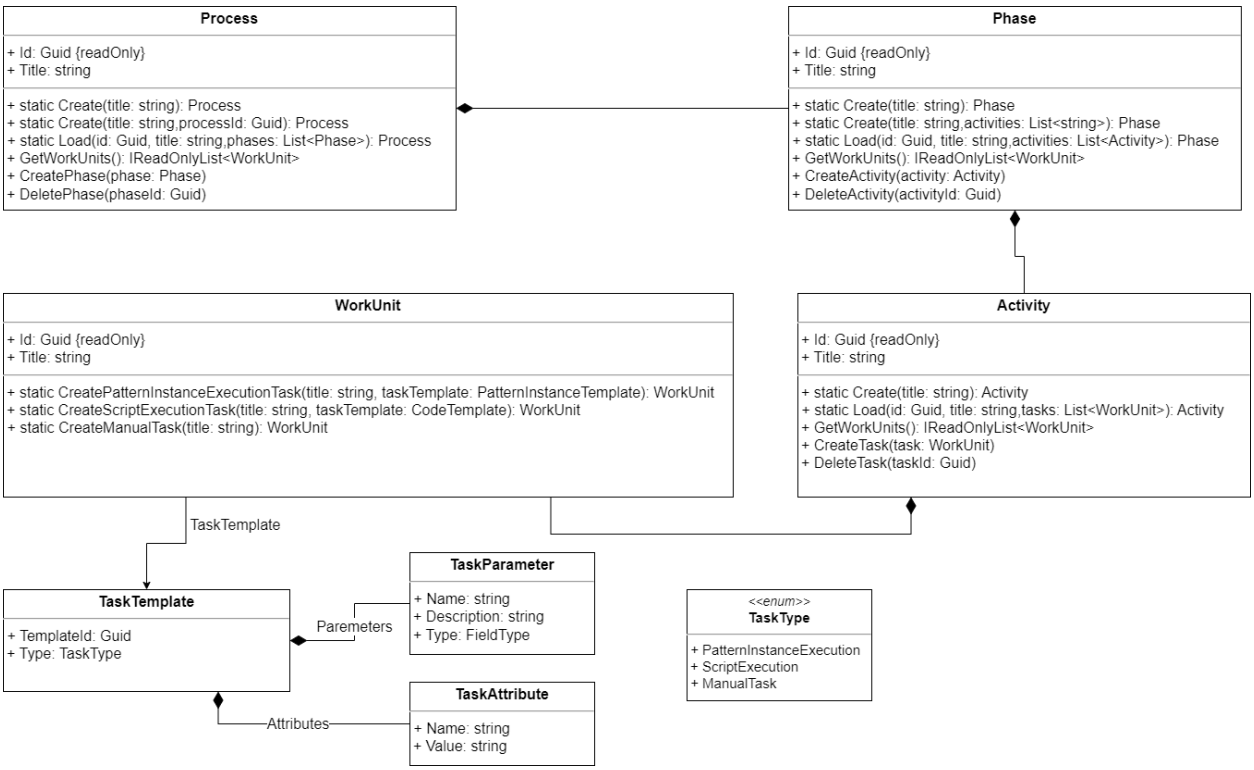

Figure 8: Process component in ModelTransformations microservice

The ModelTransformations microservice uses pre-defined artifacts for the creation of model transformation processes, allowing for the reuse of pattern instance templates at the task-definition level and transformation pipelines at the activity-definition level. It also enables the creation of new transformation patterns as stateless services using the desired language and technology stack, which can be registered in the ModelTransformations microservice and called upon during process execution. The system architecture includes API Gateway for load balancing and routing requests, and a self-registration pattern for managing the list of transformation services. Transformation patterns can be implemented as stateless Function as a Service (FaaS), scaling out in response to workload changes. The overall architecture is shown in Figure 9.

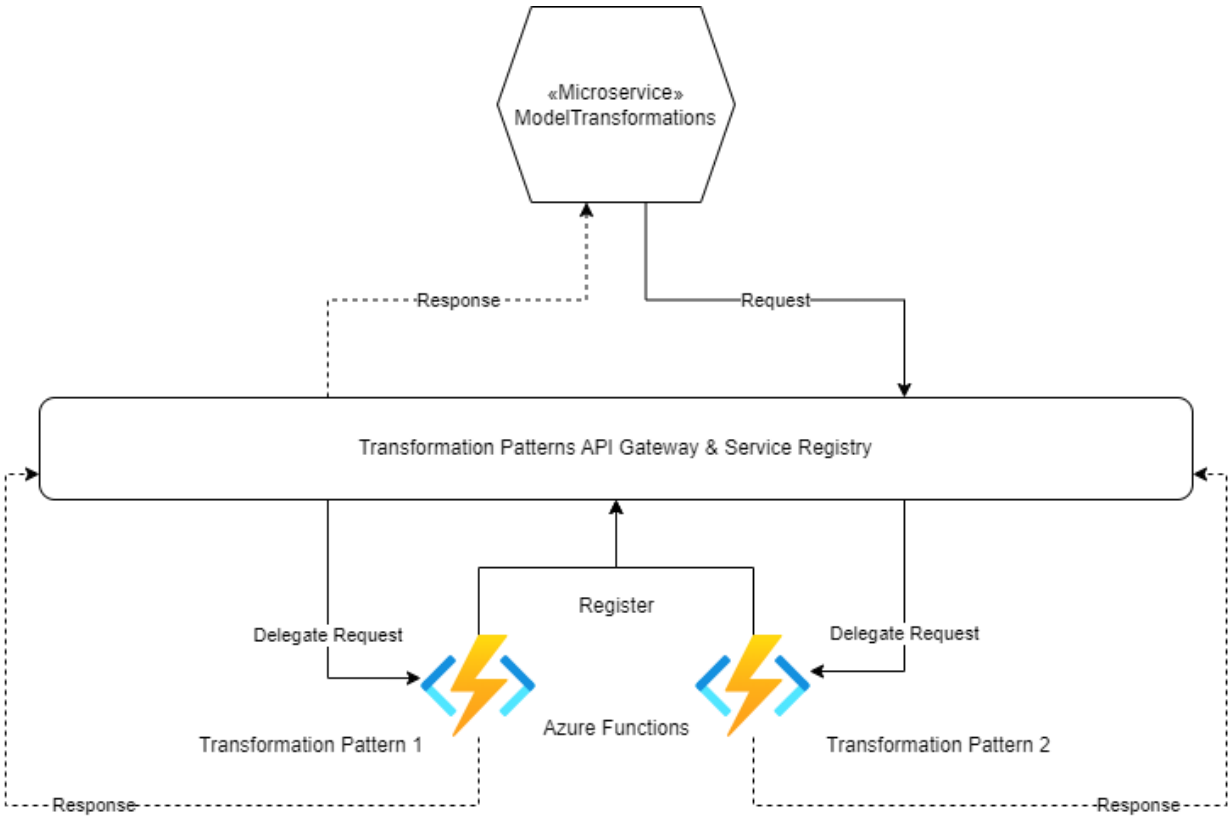

Figure 9: Extensible architecture for managing new model transformation patterns

5.6. Diagrams microservice

The Diagrams microservice converts domain object and domain concept information obtained from the DomainModels microservice into a standardized PlantUML format. The resulting diagram text is then sent to the PlantUML service to generate a graphical model diagram in PNG or SVG format. The Diagrams microservice supports Object and Class diagrams.

5.7. APIClient user interface

The APIClient project provides a user interface for accessing the modeling and transformation services offered by the MDDPlatform through a web browser. Blazor WebAssembly technology is used to create the user interface, which is downloaded to the user's browser and executed using WebAssembly. This results in fast and responsive page transitions, and once the client-side code is downloaded, the connection with the server can be terminated.

6. Criteria for Evaluating MDE Solutions

A survey study was conducted to investigate the evaluation methods of MDE solutions and the criteria and metrics used to evaluate the relevant quality attributes, especially scalability, reusability, and collaboration. Four research questions were formulated to guide the study, including: 1) how the proposed solutions related to MDE and software modeling are evaluated (RQ1), and 2) how scalability, reusability, and collaboration are evaluated in MDE approaches (RQ2, RQ3, RQ4). The following search query was used to search databases for relevant articles:

("MDD" OR "MDE" OR "Model-driven development" OR "Model-driven engineering") AND ("Reusability" OR "Scalability" OR "Collaboration" OR "Collaborative") AND ("Modeling" OR "Model"). Then, a set of criteria were defined to filter out irrelevant articles. In the first stage of filtering, papers not related to MDE were removed based on the titles and abstracts. In the second stage, the remaining papers were studied to identify those related to the four research questions. Data summarization, aggregation, and analysis were then performed to answer the research questions. The final results were used in the comparative analysis (Section 7) and the experiment (Section 8) that were conducted for evaluating the proposed approach.

6.1. Answer to RQ1

The evaluation methods used in MDD and MDE solutions can be categorized into four main groups: university case studies, industrial case studies, user studies, and comparative analysis (Shamsujjoha et al., 2021). University case studies involve creating an initial version of an application or implementing certain components on a specific platform to evaluate the solution. Qualitative criteria, user studies or industry-standard practices may be employed to assess the results (Shamsujjoha et al., 2021). Industrial case studies, on the other hand, involve creating a real version of the application as a pilot project and comparing the results with common industrial samples. Use case examples can also be defined to evaluate the impact of the solution on qualitative attributes (Shamsujjoha et al., 2021). User studies involve gathering participants' perspectives through interviews, surveys, and analysis of the collected information (Shamsujjoha et al., 2021). Comparative analysis involves creating an application or component and comparing it with other approaches (Shamsujjoha et al., 2021).

6.2. Answer to RQ2

We examined several research studies (Akiki & Maalouf, 2021; Bagherzadeh et al., 2021; Bruneliere et al., 2020; Bucchiarone et al., 2020; Daniel et al., 2019; David et al., 2023; Gómez et al., 2020; Ivanchikj et al., 2022; Jahed et al., 2021; D. Kolovos et al., 2013; D. S. Kolovos et al., 2015; López & Cuadrado, 2022; Mkaouar et al., 2023; Ogunyomi et al., 2019; Rocco et al., 2015; Tröls et al., 2022), and identified the features and requirements that are necessary to improve scalability. We also identified various quantitative and qualitative criteria for evaluating scalability. These requirements and the corresponding criteria are presented in Table 2, Table 3, and Table 4.

Table 2: Requirements and features that are necessary to fulfil scalability

Requirements & Features	Details
Support for systematic creation of large-scale models	It should be possible to define domain specific languages
	It should be possible to model the scope of the problem from different perspectives
	It should be possible to systematically manage the models resulting from the description of the problem from various perspectives.
Support for systematic development of domain-specific languages	It should be possible to define the constituent concepts of a language.
	It should be possible to determine the characteristics of a concept and its relationship with other concepts that make up the language.
Support for interaction and collaboration of large teams of designers	The model changes resulting from team member collaboration and participation should be published online.
	It should be possible to decompose the problem domain into subdomains and assign each subdomain to a group of designers.

Requirements & Features	Details
	There should be systematic management of the problem domain, its subdomains, and the models describing each subdomain.
Support for large-scale models transformation	It should be possible to perform transformations on large models incrementally.
	It should be possible to define fine-grained transformations, each of which is performed on a small part of the model.
	It should be possible to define and execute a chain of transformations on large models.
	It should be possible to perform transformations on large models without the need to fully load that model.
Support for storing, indexing, and retrieving the models efficiently	Storage should be done in such a way that it is possible to search among the elements of a model.
	It should be possible to retrieve some elements of a model without having to load other elements.
Support for the large model decomposition into the different parts	The platform should support the large model decomposition into different parts, and build and compose them again.
Support for merging different models and creating a large model	It should be possible to merge the data of several models with the same metamodel in the form of a larger model with the similar metamodel.
	It should be possible to merge the data of several models with various metamodels in the form of an output model whose metamodel is a superset of input metamodels.
Providing a simple view of models	Models should be organized in the form of a hierarchical structure of concepts and domain objects.
	It should be possible to display the details of each element according to the user's request.
Organizing the models at different levels of abstraction	It should be possible to decompose the problem domain into the subdomains and describe them from different perspectives.
	It should be possible to describe and manage models at different abstraction levels of CIM, PIM, and PSM.
Support for querying the large-scale models	The platform should support querying the large-scale models.
Support for cloud-based storage and remote access to the artifacts via the internet	The platform should support cloud-based storage and remote access to the artifacts via the internet.
Access to specific model elements with support for lazy loading as needed	It should be possible to access and display parts of a model, without the need to load it completely.
	It should be possible to load the elements of a model, as needed or based on the user's request.
Support for parallelization and concurrent execution of transformations	The platform should support parallelization and concurrent execution of transformations to improve performance.
Support for iterative incremental transformations	The platform should support iterative incremental transformations, and maintain the source and target models in sync.
Support for complexity management by defining the process of model-driven development	It should be possible to define and configure the MDD processes.
	It should be possible to run the MDD process and perform transformations in an automatic or semi-automatic way.

Table 3: Qualitative criteria for evaluating the scalability quality attribute

Domain	Criteria
Model Transformation	The ability to transform large-scale models
	The ability to execute transformation concurrently
	The ability to perform transformations in iterative incremental fashion
	The ability to maintain the source and target models in sync
	The ability to perform lazy computation
	The accuracy of the transformed models
Tools & Modeling Approaches	The ability to create models in a systematic manner
	The ease of creating large-scale models
	The accuracy of the created models
	The ability to develop languages in a systematic manner
	The ease of developing domain-specific languages
	The accuracy of the developed languages
	The ability to query large-scale models

	The ease of querying large-scale models
	The accuracy of the query results
	The ability to explore and discover models in a systematic manner
	The ability to store, index, and retrieve models efficiently
	The accuracy of the stored models
	The ability to access artifacts stored remotely
	The ease of accessing artifacts remotely
	The ability to decompose models
	The ease of decomposing models
	The accuracy of the decomposed models
	The ability to merge models
	The ease of merging models
	The accuracy of the merged models
	The ability to load model elements using lazy loading
	The ability to access specific model elements without loading entire model
	The ability to visualize models and traverse between model elements
	The ease of visualizing and traversing models
	The accuracy of the visualized models
	The ability to organize models at the different levels of abstraction
	The ease of viewing models at the different levels of abstraction
	The ability to organize model elements in a hierarchical structure
	The ease of navigating the hierarchical structure of the model elements
	The accuracy of the hierarchical structure of the model elements
MDE Process	The ability to create and refine models through collaboration of large teams of designers
	The ease of creating and refining models through collaboration of large teams of designers
	The ability to define the MDD process
	The ability to manage complexity in a systematic manner
	The ease of configuring and running the MDD processes

Table 4: Quantitative criteria for evaluating the scalability quality attribute

Domain	**Criteria**	**Influential variables**
Model Transformation	Transformation execution time	Structural complexity of the models; Size of the models; Number of elements in models; Complexity of transformation rules; The number of processors
	Memory consumption	
	Resource consumption	
	The number of transformations that can be executed concurrently	
Tools & Modeling Approaches	Model loading time	Structural complexity of the models; Size of the model; Number of elements in the models
	Model storage time	
	Model updating time	
	Time required for the change propagation	
	Time required for model comparison	
	Time required for model querying	
	Time required for model traversal	
	Maximum displayable elements of a model	
	Memory consumption	
	Resource consumption	
	Network resource consumption	
	Computational complexity	
MDE Process	Number of concurrent users in the modeling process	Geographical distribution of team members
	Time required for receiving the change notifications in the collaborative modeling.	

6.3. Answer to RQ3

After reviewing several studies (Alam et al., 2018; Chavarriaga et al., 2014; Cuadrado et al., 2014; Franzago et al., 2018; Kusel et al., 2015; Panahandeh et al., 2021; Wagelaar et al., 2011; Wimmer et al., 2010), we identified several requirements and features that are necessary to support

reusability. Additionally, we identified a set of qualitative and quantitative criteria and metrics related to reusability that can be used to assess this quality attribute. Table 5 and Table 6 present these requirements and the corresponding criteria for evaluating reusability.

Table 5: Requirements and features that are needed to support reusability

Requirements & Features	**Details**
Support for defining modular model transformations	It should be possible to define fine-grained transformations
	It should be possible to define a chain of transformations and run this chain automatically or semi-automatically.
Support for reusing the existing transformations	The platform should support reusing the existing transformations.
Support for composing transformations and creating a chain of transformations	The platform should support composing the existing transformations and creating a chain of transformations.
Support for reusing existing models	It should be possible to use the elements defined in a domain model to create a model with a similar metamodel, but not necessarily in the same domain.
	It should be possible to automatically use part of the model elements to create a model with a different metamodel.
Support for reusing existing metamodels	It should be possible to query and view the existing modeling languages (metamodels) and their constituent elements.
	It should be possible to reuse the constituent concepts of a metamodel to define a new metamodel.
Support for the abstract mechanisms for applying transformations to a variety of metamodels	Dependencies between the transformation and their input and output metamodels should be loosely defined.
	It should be possible to define a transformations in the form of transformation model.
	Model transformations must be configurable to conform to different metamodels.
	Model transformations must be defined in the form of the template or pattern that can be used for the family of metamodels.
Support for tailoring model transformations to a specific metamodel	Transformations should be parametric and configurable at run time.
	To define a transformation, the transformation model approach or higher-order transformations can be used so that these transformations can be transformed into the desired transformations, suitable for a specific metamodel.
Support for reusing the existing MDD processes	It should be possible to define and query the existing MDD processes.
	It should be possible to configure the existing MDD process according to the new problem domain.
Support for the well-defined interfaces to describe and retrieve model elements	The necessary APIs for creating, querying and retrieving model elements must be defined.
	It should be possible to access the elements remotely by calling the web API
Support for discovering artifacts created through collaboration during different phases	It should be possible to query among the existing models based on the metamodel
	It should be possible to query among existing models based on the domain and abstraction levels of CIM, PIM and PSM.
	It should be possible to query transformations based on input and output metamodels.
	It should be possible to query transformations based on the abstraction level of input and output models.

Table 6: Qualitative criteria for evaluating the reusability quality attribute

Domain	**Criteria**
Model Transformation	The ability to define modular model transformations
	The ease of defining modular model transformations
	The ability to use model transformations across different metamodels
	The ease of using model transformations across different metamodels
	The ability to tailor model transformations to a specific metamodel
	The ease of tailoring model transformations to a specific metamodel
	The ability to compose and create the chain of model transformations
	The ease of composing and chaining model transformations
	The ability to search model transformations
	The ease of searching model transformations

	The accuracy of the search results
	The ability to reuse existing model transformations
	The ease of reusing existing model transformations
Tools & Modeling Approaches	The ability to reuse existing models
	The ease of reusing existing models
	The accuracy of the reused models in the new context
	The ability to reuse existing metamodels
	The ease of reusing existing metamodels
	The ability to discover artifacts created through collaboration during different phases
	The ease of discovering artifacts created through collaboration during different phases
	The ability to describe and retrieve the model specifications
	The ease of describing and retrieving the model specifications
	The accuracy of the retrieved specification
MDE Process	The ability to reuse existing MDD processes
	The ease of reusing existing MDD processes
	The accuracy of the reused MDD processes in the new context

The modularity of transformation rules is one of the evaluation metrics that measures how easily these rules can be decomposed or merged and is closely related to the quality attributes of flexibility, ease of change, and reusability (Panahandeh et al., 2021). The following formula can be used to quantify this metric (Chavarriaga et al., 2014):

$$m=1-(d/r)$$

Here, the d value represents the number of dependencies (implicit or explicit) between transformation rules, and r represents the number of transformation rules in the transformation design model (TDM) (Chavarriaga et al., 2014).

6.4. Answer to RQ4

By reviewing existing works (Akiki & Maalouf, 2021; Alam et al., 2018; Franzago et al., 2018; D. Kolovos et al., 2013), we have identified a set of requirements and features that facilitate the collaboration and interaction of the teams during the modeling process. We have also indentified qualitative and quantitative criteria and metrics for evaluating this quality attribute. Table 7 and Table 8 outline these requirements and criteria.

Table 7: Requirements and features that facilitate collaboration

Requirements & Features	**Details**
Support for multi-user environment with online or offline collaboration capability	The platform should support a multi-user environment with online or offline collaboration capability.
Providing an infrastructure for managing models throughout their lifecycle	The platform should provide an infrastructure for managing models throughout their lifecycle.
Providing a repository for storing models and related metadata	The platform should provide a repository for storing models and related metadata.
Providing modeling tools for creating, editing, and deleting models	The platform should provide modeling tools for creating, editing, and deleting models.
Ability to share models in different formats	The platform should have the ability to share models in different formats.
Support for a version control system	It should be possible to store artifacts in different versions.
	It should be possible to record and update the history of changes.
	It should be possible to search among different versions of artifacts.
Support for model merging	The platform should be able to merge the models.
Support for conflict management	It should be possible to synchronize and publish changes of models as a result of team collaboration.

Requirements & Features	Details
	It should be possible to manage the conflict created in the model due to simultaneous editing.
	It should provide facilities for resolving conflicts
	It should be possible to highlight the selected element in the model by one of the stakeholders when modeling online.
	It should be possible to lock elements and highlight the locked element by other stakeholders.
	It should be possible to assign colors to each user when modeling online.
	It should be possible to display the mouse pointer of other stakeholders when modeling online.
Support for visualization of models	It should be possible to illustrate the changes applied to the models in the editor during online modeling.
Collecting information related to various aspects of the project	The platform should collect information related to various aspects of the project.
Support for a public notification service	The platform should support a public notification service and notify the model changes
Ability to monitor the stakeholders' activities during online modeling	It should be possible to display the details of the updating operation.
	It should be possible to display the list of active stakeholders at the moment.
Ability to interact in a defined and traceable manner with the artifacts	It should be possible to find out who is doing what on which part of the artifact.
	It should be possible to track the topics raised in connection with the artifacts.
	It should be possible to share design decisions.
	It should be possible to communicate indirectly through sending documents, comments and explanations.

Table 8: Criteria for qualitative analysis of collaboration

Domain	Criteria
Tools & Modeling Approaches	The ability to collaborate in a multi-user environment
	The ease of online or offline collaboration in a multi-user environment
	The ability to manage models throughout their lifecycle
	The ease of managing models throughout their lifecycle
	The ability to store models and their corresponding metadata in a central repository
	The accuracy of the stored models
	The ability to manage and manipulate the models by the tool
	The ease of using modeling tools to manage and manipulate the models
	The ability to share models in different format
	The ease of sharing models
	The accuracy of the shared models
	The ability to manage different version of models
	The ease of managing artifacts in different version
	The ability to merge models
	The ease of merging models
	The accuracy of the merged models
	The ability to manage conflicts in a systematic manner
	The ease of managing conflicts
	The accuracy of the automatic conflict management
	The ability to visualize models
	The ease of visualizing models
	The accuracy of the visualized models
MDE Process	The ability to collect project information from different aspect
	The ability to notify model changes
	The ability to monitor activities during collaborative modeling
	The ease of monitoring activities
	The ability to trace model manipulation to the user interaction
	The ease of interaction with models

To effectively collaborate in the modeling process, it is crucial to track team members' participation and the changes that they have made to the models. CHECKSUM (Akiki & Maalouf, 2021) is a technique that enables the monitoring of collaborative modeling, tracking of changes, and maintenance of an immutable Changelog. The Changelog is used to measure the level of contribution, which can be evaluated from three perspectives: temporal, operational, and qualitative. The temporal perspective measures the time spent on various operations performed on the model. The operational perspective indicates the type of operation performed on the model, while the qualitative perspective assesses the user's opinion regarding the output of their work. To quantify the level of contribution, each dimension can be assigned a weight and score, and the overall contribution score can be calculated by combining them.

7. Comparative Analysis

In this section, we will analyze MDDPlatform and other modeling services such as WebGME, AToMPM, MORSE, and MDEForge using qualitative criteria related to scalability, reusability, and collaboration, and thereby compare the level of support that they provide (fully support, partially support, and no support) for the features associated with these criteria.

7.1. Comparison based on scalability evaluation criteria

Table 9 shows the results of evaluating and comparing the proposed approach and other services from the perspective of scalability. As seen in this table, MDDPlatform is superior to its counterparts in its support for most of the requirements and features pertaining to scalability.

Table 9: Evaluation and comparison from the perspective of scalability

Requirements & Features	WebGME	MORSE	AToMPM	MDEForge	MDDPlatform
Support for systematic creation of large-scale models	●	○	●	○	●
Support for systematic development of domain-specific languages	●	○	●	○	●
Support for interaction and collaboration of large teams of designers	○	○	◐	○	●
Support for large-scale models transformation	○	○	-	-	◐
Support for storing, indexing, and retrieving the models efficiently	●	●	◐	◐	●
Support for the large model decomposition into the different parts	○	○	○	○	○
Support for merging different models and creating a large model	○	○	○	○	◐
Providing a simple view of models	●	○	◐	-	●
Organizing the models at different levels of abstraction	○	○	○	○	●
Support for querying the large-scale models	○	◐	○	○	○
Support for cloud storage and remote access to the artifacts via the internet	●	●	◐	●	●
Access to specific model elements with support for lazy loading as needed	○	◐	○	○	●
Support for parallelization and concurrent execution of transformations	○	○	○	○	○
Support for iterative incremental transformations	○	◐	-	●	●
Support for complexity management by defining the process of MDD	○	◐	○	○	●
○: No support; ◐: Partially Support; ● : Fully support					

7.2. Comparison based on reusability evaluation criteria

Table 10 shows the results of evaluating and comparing the proposed approach and other services from the reusability point of view. As seen in this table, MDDPlatform is superior to its counterparts in its support for most of the requirements and features pertaining to reusability.

Table 10: Evaluation and comparison from the reusability point of view

Requirements & Features	WebGME	MORSE	AToMPM	MDEForge	MDDPlatform
Support for defining modular model transformations	◐	○	○	○	◐
Support for reusing the existing transformations	◐	○	○	●	●
Support for composing transformations and creating a chain of transformations	○	○	-	●	●
Support for reusing existing models	◐	◐	◐	◐	●
Support for reusing existing metamodels	-	-	-	-	●
Support for the abstract mechanisms for applying transformations to a variety of metamodels	○	○	○	◐	◐
Support for tailoring model transformations to a specific metamodel	○	○	○	◐	◐
Support for reusing the existing MDD processes	○	○	○	○	●
Support for the well-defined interfaces to describe and retrieve model elements	●	●	○	○	●
Support for query about existing models based on metamodel	○	○	○	●	●
Support for query about the existing models based on the domain and abstraction levels of CIM, PIM and PSM.	○	○	○	○	●
Support for query about transformations based on input and output metamodels.	○	○	○	●	●
Support for query about transformations based on the abstraction level of input and output models.	○	○	○	○	●
○: No support; ◐: Partially Support; ●: Fully support					

7.3. Comparison based on collaboration evaluation criteria

Table 11 shows the results of the evaluation and comparison of the proposed approach and other services from the perspective of collaboration. As seen in this table, MDDPlatform is either superior to or at the same level as its counterparts in its support for many of the requirements and features pertaining to collaboration; however, the need still remains for significant progress in this regard.

Table 11: Evaluation and comparison from the perspective of collaboration

Requirements & Features	WebGME	MORSE	AToMPM	MDEForge	MDDPlatform
Support for multi-user environment with online or offline collaboration capability	○	○	●	○	●
Providing an infrastructure for managing models throughout their lifecycle	●	●	●	●	●
Providing a repository for storing models and related metadata	●	●	●	●	●
Providing modeling tools for creating, editing, and deleting models	●	●	●	◐	●
Ability to share models in different formats	○	○	◐	◐	◐
Support for a version control system	-	●	○	-	○
Support for model merging	○	○	○	○	◐
Support for conflict management	○	○	○	○	◐
Support for visualization of models	◐	○	●	◐	◐

Collecting information related to various aspects of the project	○	○	○	○	○
Support for a public notification service	○	○	○	○	○
Ability to monitor the stakeholders' activities during online modeling	○	○	○	○	◐
Ability to interact in a defined and traceable manner with the artifacts	○	○	○	○	○
○: No support; ◐: Partially Support; ●: Fully support					

8. Case Study and Experiment

The objective of the case study was to assess the effectiveness of the proposed approach in enhancing scalability, reusability, and collaboration in MDD. To achieve this, an experiment was conducted where participants were tasked with implementing a portion of an online ordering system (orders management) using the proposed approach and MDDPlatform. The demographic information of the participants has been presented in Appendix A. To ensure consistency in understanding the problem, the system was initially analyzed using the CRAC method and the output was presented as a CRAC model to all participants. The experiment aimed to evaluate if the proposed approach and MDDPlatform could aid in managing complexity, improving artifact reuse, enhancing scalability, and facilitating collaboration and participation in the modeling process. After completing the task, participants were asked to complete a questionnaire where they could express their opinions on each statement by selecting one of five options: completely agree, agree, not sure, disagree, or strongly disagree.

8.1. Experiment specification

The experiment involved multiple stages and steps, which were outlined in a written document provided to participants. Additionally, we recorded 22 videos to introduce the MDDPlatform and guide users through the experiment process. The experiment consisted of five main steps, including platform setup, loading predefined packages, defining the problem domain and creating domain models, configuring and executing the model-driven process, and code generation. At the end of the experiment, participants were asked to complete two online questionnaires. The first questionnaire had 16 statements related to scalability, reusability, and collaboration attributes, while the second questionnaire had 11 statements related to ease of use and usefulness of the tool. The statements for both questionnaires can be found in Table 12 and Table 13.

Detailed information on the particulars of the experiment has been provided in the Appendices file that has been made available on Mendeley Data (Vahdati & Ramsin, 2023).

Table 12: First questionnaire: evaluation as to scalability, reusability and collaboration

No.	Statement
1	Deploying MDDPlatform on the internet, without the need for installation on a local machine, facilitates remote data access and collaboration among distributed team members.
2	The ability to load specific domain languages, predefined transformation patterns, and processes in MDDPlatform facilitates reuse of previous artifacts and reduces the initial cost of adopting a model-driven approach.
3	The ability to search for specific domain languages and models based on them in MDDPlatform improves finding and reusing previous artifacts.
4	Breaking down the problem domain into subdomains and creating models at different abstraction levels helps manage complexity in MDDPlatform.

5	Breaking down the problem domain into subdomains and creating models at different abstraction levels enables systematic description of the problem domain in MDDPlatform.
6	Decomposing the problem domain into subdomains and assigning each subdomain to a group of modelers can serve as a basis for design and modeling task assignments and improve teamwork and collaboration in MDDPlatform.
7	In MDDPlatform, the ability to create multiple granular models at a specific abstraction level that describe the problem from different perspectives helps manage complexity both at the metamodel definition level and at the model creation and description level.
8	Providing model transformations in the form of parametric patterns with the ability to instantiate and configure them in MDDPlatform improves reuse of transformations.
9	The ability to define a model-driven process and its automatic and semi-automatic execution in MDDPlatform helps manage complexity and guide the execution of model transformations at different levels compared to the ad hoc approach.
10	The ability to define and configure a model-driven process in MDDPlatform facilitates reuse of design and architecture patterns for other projects.
11	The ability to define a chain of transformations in the form of a pipeline or high-level process in MDDPlatform provides the possibility of applying transformations incrementally.
12	In MDDPlatform, the ability to define manual tasks during the MDD process or model transformation pipeline facilitates interrupting the transformation process and reviewing and revising the transformation results up to that point.
13	In MDDPlatform, it is possible to define multiple configurations for a MDD process. By defining a configuration for each subdomain of the problem, it is possible to collaborate among different teams and run the MDD processes concurrently for different subdomains.
14	Providing model specification in a hierarchical structure (model concepts, instances of each concept, property values of each instance, and the relations among them) enables lazy loading of information and improves scalability regarding large models in MDDPlatform.
15	When describing a model in MDDPlatform, if a member of the modeling team changes the model, these changes are simultaneously propagated to other members working on that model. This improves collaboration capabilities.
16	In MDDPlatform, the ability to automatically merge information from one model into another model with the same or different metamodel provides the possibility of reusing all or part of existing model data.

Table 13: Second questionnaire: evaluating the usefulness and ease of use of the MDDPlatform tool

No.	Statements
1	Using MDDPlatform accelerates modeling, model transformation, and code generation tasks.
2	Productivity increases with the help of MDDPlatform.
3	The proposed approach in MDDPlatform is an effective method for MDD
4	The MDD process is facilitated using MDDPlatform.
5	MDDPlatform is a useful and practical tool for MDD.
6	It is easy to learn MDDPlatform.
7	The way to use MDDPlatform is understandable and transparent.
8	MDDPlatform is flexible.
9	The MDDPlatform tool can be controlled by the user.
10	MDDPlatform is easy to use.
11	It is easy to gain skills and experience in MDDPlatform.

8.2. Experiment results

The answers of the participants to the questions in the first and second questionnaires have been shown in Appendix B and Appendix C, respectively. Likert scales were used to assign weights to the five alternative options that convey the opinion of participants about the statements in the questionnaires: Strongly agree=5, Agree = 4, Not sure = 3, Disagree = 2, and Strongly Disagree = 1. In the first questionnaire, we seek to evaluate the impact of MDDPlatform on the scalability, reusability, and collaboration from the MDD perspective. We categorize the statements of this questionnaire into three groups: The statements that evaluate the impact of MDDPlatform on the

scalability (Q4, Q5, Q7, Q9, Q11, Q14), the statements that evaluate the impact of MDDPlatform on the reusability (Q2, Q3, Q8, Q10, Q16), and the statements that evaluate the impact of MDDPlatform on the collaboration (Q1, Q6, Q12, Q13, Q15). We define three statistics based on these categories including scalability average score, reusability average score, and collaboration average score. For each participant, these statistics are calculated according these formulas:

$$Scalability.average.score_i = \frac{Q_{i,4} + Q_{i,5} + Q_{i,7} + Q_{i,9} + Q_{i,11} + Q_{i,14}}{6}$$

$$Reusability.average.score_i = \frac{Q_{i,2} + Q_{i,3} + Q_{i,8} + Q_{i,10} + Q_{i,16}}{5}$$

$$Collaboration.average.score_i = \frac{Q_{i,1} + Q_{i,6} + Q_{i,12} + Q_{i,13} + Q_{i,15}}{5}$$

$$Q_{i,j} = The\ response\ of\ participant\ i\ to\ the\ Statement\ j\ (Q_j)$$

Figure 10 shows the statistical information of the answers to the first questionnaire. We aggregated the responses and presented the percentage of votes received for each option, corresponding to each statement in the first questionnaire. Figure 11 shows the cumulative result. For each participant, the average scores of statements corresponding to scalability, reusability, and collaboration are calculated and shown in the Figure 12, Figure 13, and Figure 14.

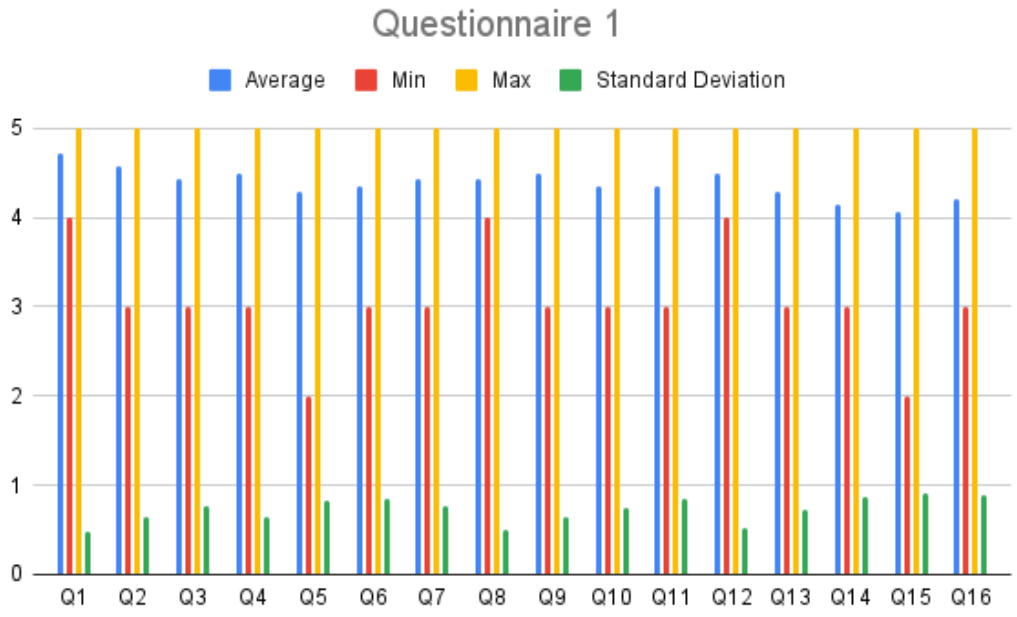

Figure 10: Statistical information of questionnaire 1

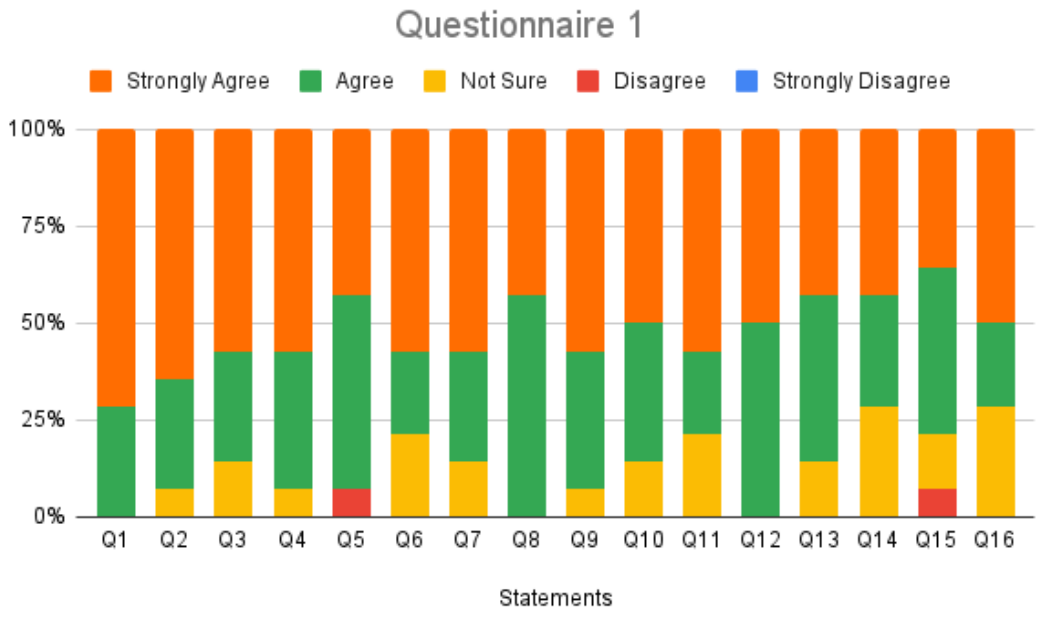

Figure 11: Aggregation of responses in the first questionnaire

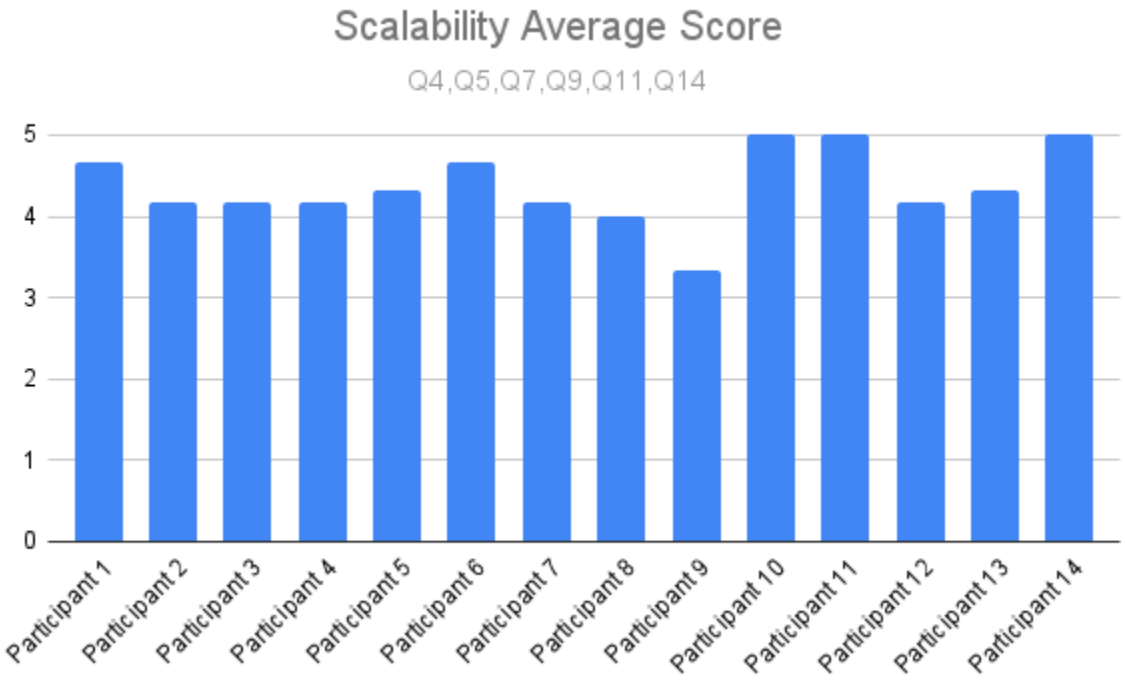

Figure 12: Scalability average score

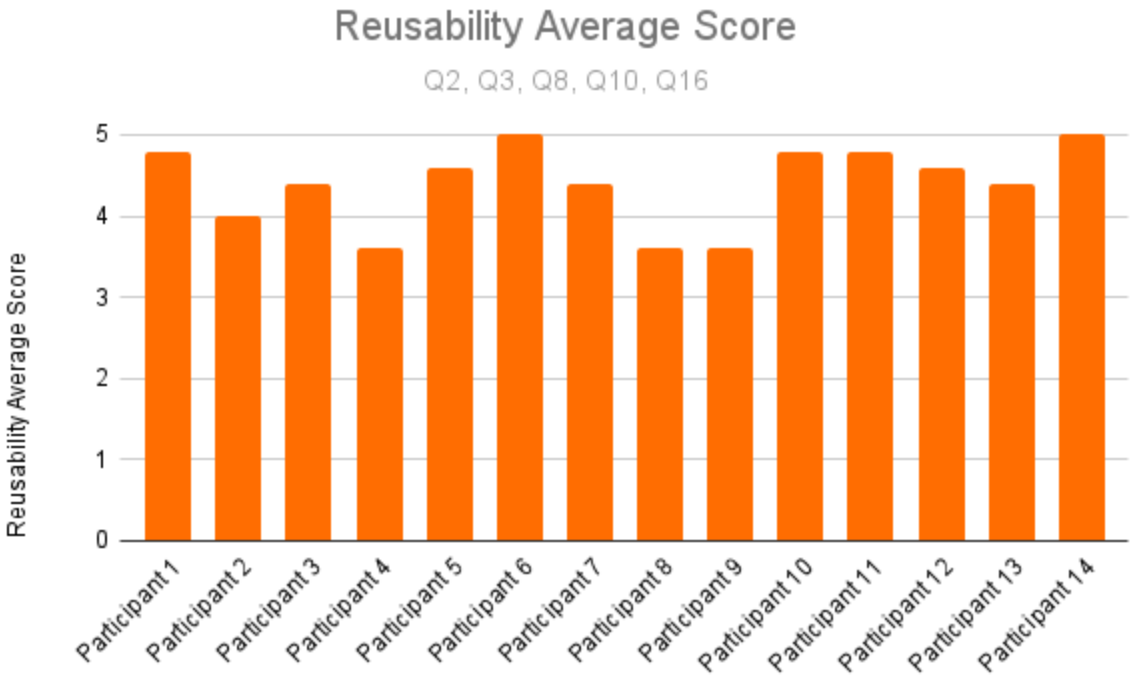

Figure 13: Reusability average score

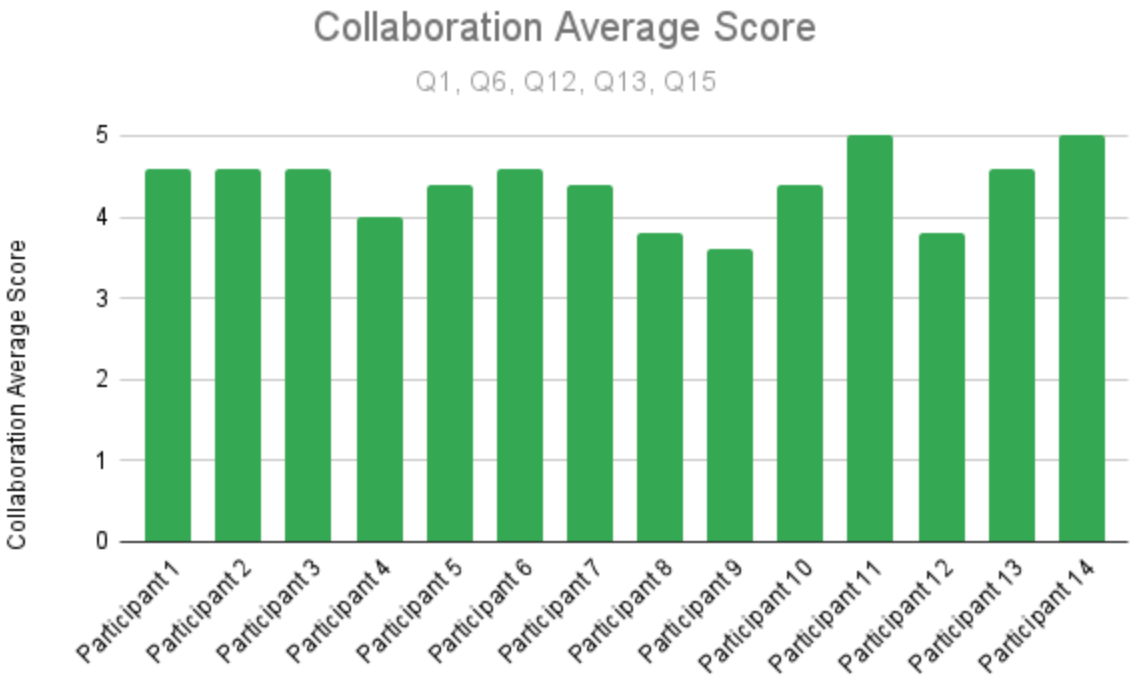

Figure 14: Collaboration average score

In the second questionnaire, we evaluate the usability and ease of use of the MDDPlatform. The statements of this questionnaire are divided into two category: The statements that examine the

usability of MDDPlatform (Q1, Q2, Q3, Q4, Q5, Q6), and the statements that examine the MDDPlatform ease of use (Q7, Q8, Q9, Q10, Q11). We define two statistics based on these categories including usability average score, and ease of use average score. For each participant, these statistics are calculated according these formulas:

$$Usability.average.score_i = \frac{Q_{i,1} + Q_{i,2} + Q_{i,3} + Q_{i,4} + Q_{i,5} + Q_{i,6}}{6}$$

$$EaseOfUse.average.score_i = \frac{Q_{i,7} + Q_{i,8} + Q_{i,9} + Q_{i,10} + Q_{i,11}}{5}$$

The statistical information of the second questionnaire responses and cumulative result are presented in Figure 15 and Figure 16.

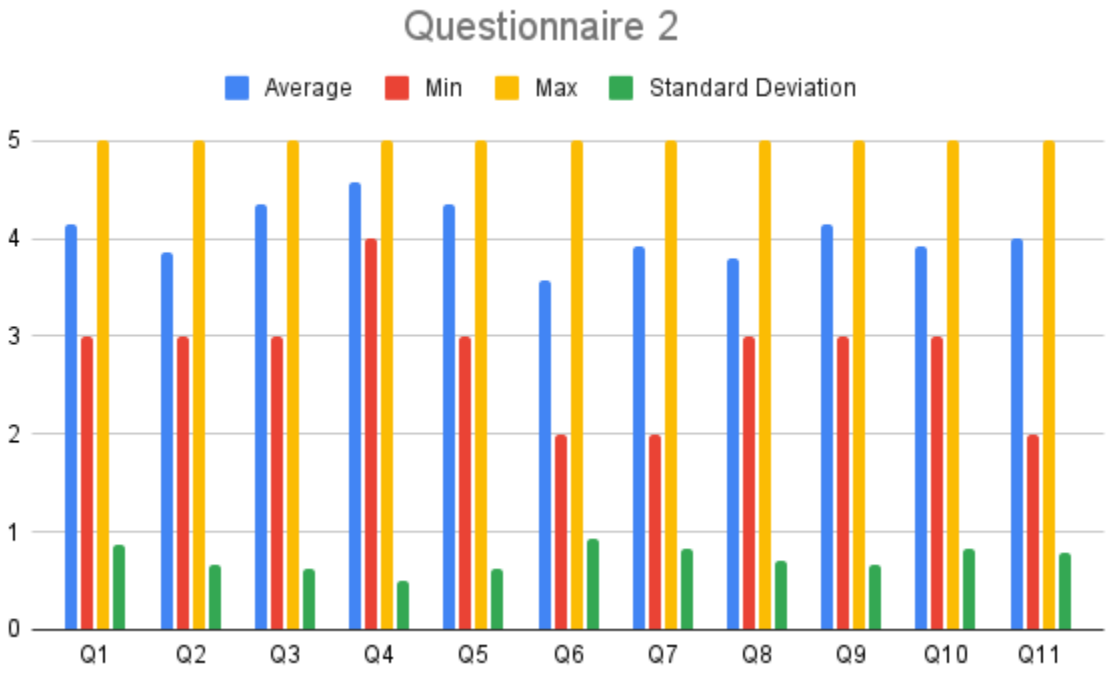

Figure 15: Statistical information of questionnaire 2

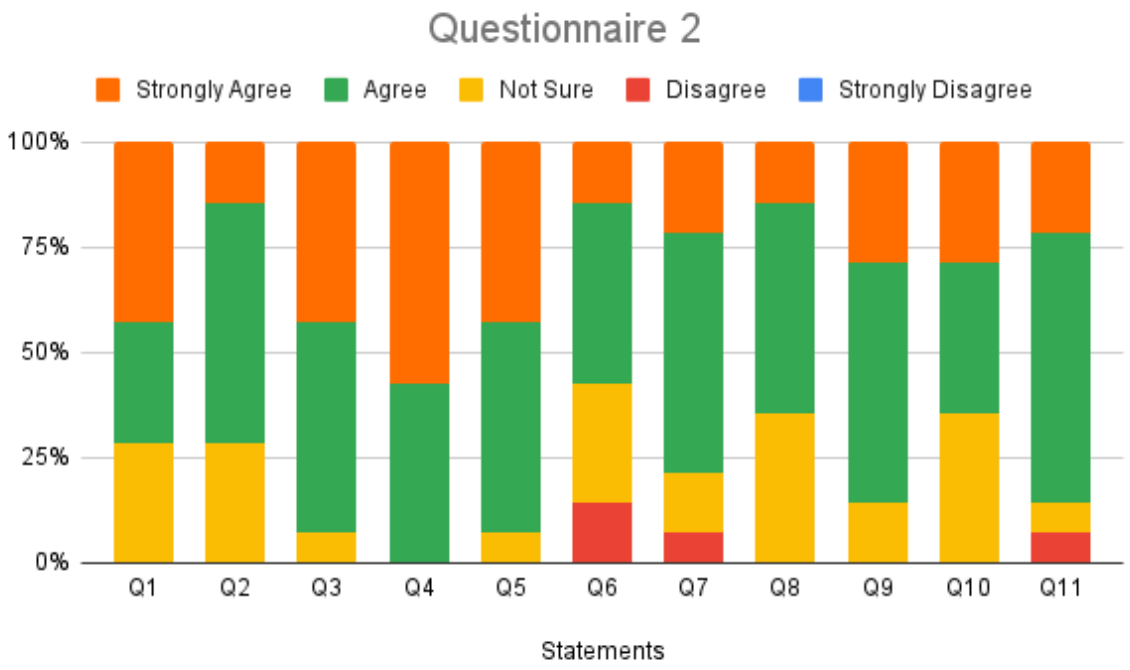

Figure 16: Aggregation of responses in the second questionnaire

The average score of statements related to usability and ease of use quality attributes were calculated for each participant and are presented in Figure 17 and Figure 18.

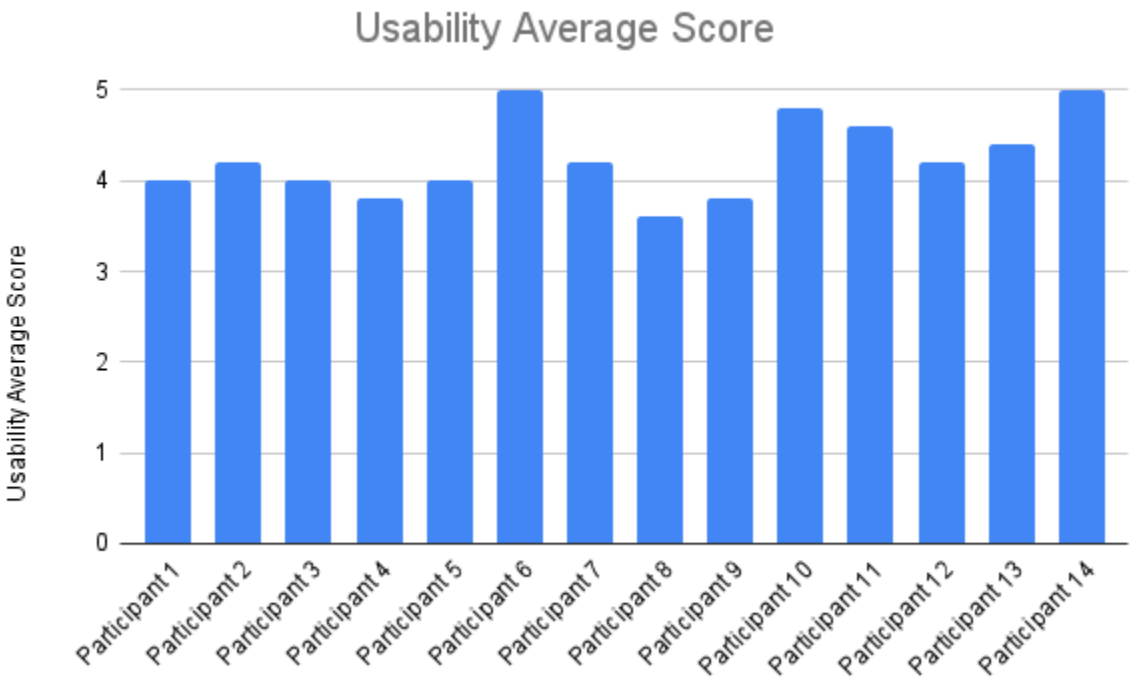

Figure 17: Usability average score

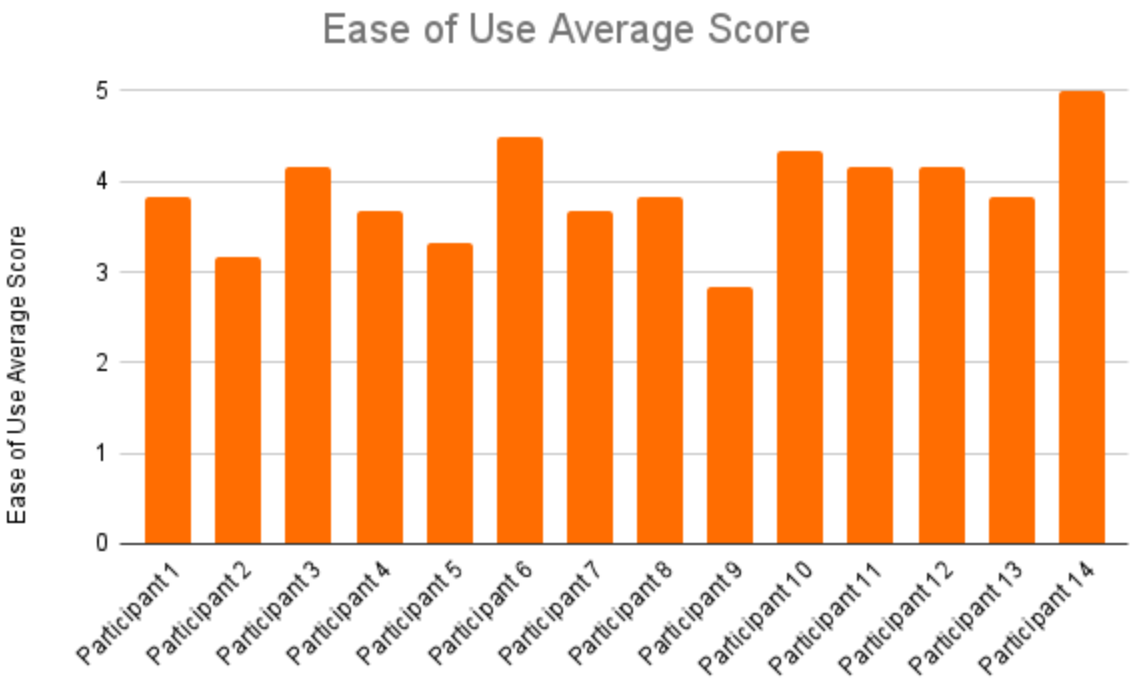

Figure 18: Ease of use average score

Table 14: Scalability, Reusability, Collaboration, Usability, and Ease of Use Average Score

Participant	Scalability Average Score	Reusability Average Score	Collaboration Average Score	Usability Average Score	Ease of Use Average Score
1	4.67	4.8	4.6	4	3.83
2	4.17	4	4.6	4.2	3.17
3	4.17	4.4	4.6	4	4.17
4	4.17	3.6	4	3.8	3.67
5	4.33	4.6	4.4	4	3.33
6	4.67	5	4.6	5	4.5
7	4.17	4.4	4.4	4.2	3.67
8	4	3.6	3.8	3.6	3.83
9	3.33	3.6	3.6	3.8	2.83
10	5	4.8	4.4	4.8	4.33
11	5	4.8	5	4.6	4.17
12	4.17	4.6	3.8	4.2	4.17
13	4.33	4.4	4.6	4.4	3.83
14	5	5	5	5	5
Average	4.37	4.4	4.39	4.26	3.89
Min	3.33	3.6	3.6	3.6	2.83
Max	5	5	5	5	5
SD	0.46	0.51	0.43	0.45	0.56

The results of the first questionnaire and Figure 12, Figure 13, and Figure 14 show that most participants in this experiment believe that the approach and features provided by MDDPlatform

have improved scalability, reusability, and collaboration. The results of calculating the three statistics Scalability.average.score, Reusability.average.score, and Collaboration.average.score for each participant in this case study are shown in Table 14. The average value of these criteria for the quality attributes of scalability, reusability, and collaboration are 4.37, 4.4, and 4.39 respectively, indicating that most participants in this experiment agree or strongly agree that MDDPlatform has improved these quality attributes.

The results of the second questionnaire in Figure 17 show that most participants in this experiment agree that MDDPlatform is a useful tool for MDD approaches. The result of calculating the statistic Useability.average.score for each participant in Table 14 and the average value of 4.26 also confirms this issue. Regarding ease of use in Figure 18, the statistic EaseOfUse.average.score related to this attribute has obtained an average score of 3.89 among all participants, which shows weaker performance compared to other quality attributes. One reason for this issue may be that instead of deploying MDDPlatform on the web, we provided them with Docker images to run MDDPlatform locally, and one of the participants in this experiment faced challenges in setting up this platform.

Detailed information on the results of the experiment has been provided in the Appendices file that has been made available on Mendeley Data (Vahdati & Ramsin, 2023).

9. Threats to Validity

In this section, we outline the potential threats to the validity of our research and the steps that we have taken to address them. To ensure the accuracy of our findings, we enlisted participants from diverse backgrounds to take part in the experiment and evaluation, and implemented measures to reduce external threats to validity. We also employed various data collection and analysis methods. To enhance construct validity, we conducted feasibility analysis and experiments with the help of experienced individuals, identified and resolved any uncertainties in the questions, and ensured internal validity by auditing and maintaining a clear chain of reasoning. We aimed to improve external validity by inviting participants from different industries and backgrounds. Additionally, we ensured reliability and repeatability by carefully defining the experiment process, seeking expert feedback on the experiment definition, and providing consistent resources to all participants. We followed guidance from Robson and McCartan (Robson & McCartan, 2017) to mitigate threats to validity, and in the following section, we provide further details on these measures.

Construct Validity

We started the study based on an initial plan and refined it during the case study. We recorded all intermediate results of the case study design and experiment in various versions of reports. To assess the risks and ensure the practicality of the case study and experiment, meetings were arranged with experienced individuals from academia and industry. A total of seven meetings were held, with a group of three PhD students and four professional members of the industry attending. The MDDPlatform and its services were introduced, and the purpose of the experiment was

explained. The experiment process, its challenges, and the designed questionnaires were also discussed, with the aim of answering several questions. We applied the feedback of the experts in the questionnaires, revised the initial plan, and informed the proposers of the changes via email.

We uploaded all the necessary data for conducting the experiment, including a textual description of the experiment, a video tutorial, and Docker images for running MDDPlatform, to Google Drive. Participants have access to this data. The questionnaire was designed online using Google Forms, and we have each participant's response and a summary of the results available.

To ensure diversity, we invited 35 people from various backgrounds in industry and academia to participate in our experiment and case study. Out of these 35 people, 14 expressed their interest in participating. After completing the experiment, they filled out questionnaires and sent their answers. We collected data through questionnaires with specific statements and also asked for their opinions on open-ended questions.

Internal Validity

One of the risks that threatened internal validity was participant fatigue, so the steps of experiments were defined in fine-grained detail to enable completion within 10-15 minutes. Participants could also complete the experiment over several days and complete the questionnaire gradually.

We provided participants with the recorded videos of the experiment's stages and steps to ensure there was no ambiguity about how the experiment should be conducted. Additionally, before conducting the experiment, we held a session with each participant to provide necessary explanations to reduce any potential ambiguity.

To ensure validity of the interpretation, we conducted a survey to identify key features and requirements for scalability, reusability, and collaboration. We then created questionnaires based on these factors to measure the impact of MDDPlatform on these quality attributes. By tracking the questionnaire responses to the requirements and features, we can evaluate MDDPlatform as to scalability, reusability, and collaboration.

To mitigate researcher bias, we implemented a member checking process. We sent each participant's response along with our interpretation and analysis of their response as a report and asked for their feedback on the interpretation and analysis presented.

We have calculated Cronbach's α to check the internal consistency of responses. Table 15 shows this measure for the first questionnaire (three average scores for scalability, reusability and collaboration) and the second questionnaire (two average scores for usability and ease of use). For all cases, the measure is greater than 0.8.

Table 15: Cronbach's α statistical evaluation

	Questionnaire 1	**Scalability, Reusability, Collaboration Average Scores**	**Questionnaire 2**	**Usability and Ease of Use Average Scores**
Cronbach's α	0.871	0.908	0.855	0.826

External Validity

We used the triangulation technique to minimize the threats of external validity. Although we invited individuals with different backgrounds to participate in the experiment and evaluation, we ultimately had to conduct this experiment with individuals who were interested in participating. Furthermore, we employed two approaches to evaluate the proposed solution: a comparative analysis and a case study.

Reliability

Before starting the experiment and case study, the initial plan and the questionnaires were reviewed in the presence of experts in order to resolve the ambiguities and make sure that the questionnaires are in line with the research objectives. Also, all the intermediate results and necessary documentation for the experiments and analyses were reviewed by the second author. By storing the experiment's definition documents, tutorial videos, and MDDPlatform Docker images (as detailed in (Vahdati & Ramsin, 2023)), it is possible to repeat it.

10. Conclusions and Future Work

We have proposed the Model-Driven Onion Architecture and have defined a new dimension for relationships between elements belonging to different models. We have also implemented a model-driven platform (MDDPlatform), which realizes the notions of *multi-level modeling as a service* and *model transformation as a service*. By defining model transformations in the form of parametric patterns and offering them as services, we aimed to improve the reusability of model transformations. The ability to define and manage models at different levels of abstraction, as well as defining and configuring the MDD processes in MDDPlatform, helped manage complexity and improve scalability. By providing MDDPlatform in the form of Docker Images, we made it possible to deploy the platform on the internet, which improves remote data access and collaboration. Additionally, by publishing changes online, collaborative modeling was facilitated for teams of modelers. To evaluate the proposed approach and tools, we conducted a comparative analysis and a case study with participants from industry and academia. The results of these evaluations have shown that MDDPlatform is a useful and superior tool as to its support for MDD and its positive effect on reusability, scalability, and collaboration.

Our future research will primarily focus on enhancing MDDPlarform as to its support for various quality attributes. We also aim to conduct industrial-scale case studies on the proposed approach.

Appendix A

The demographic information of the experiment's participants is shown in Table 16.

Table 16: Demographic information of participants

Participant No.	Academic Degree	Industry Experience (Years)	Current Status	Number of Projects Participated In	Level of Familiarity and Experience					
					Modeling & Modeling Language	MDE & MDD	Model Transformations Languages	Modeling Tools	Software Development Methodologies	Design Patterns & Architecture
1	MSc	15	Professional	5-7	4	2	2	2	4	4
2	MSc	5	Academic	5-7	5	5	5	5	4	4
3	MSc	6	Professional	3-4	3	1	1	3	4	4
4	PhD	2	Professional	8-10	4	3	3	2	5	3
5	PhD	3	Academic	3-4	5	4	4	4	5	4
6	BSc	5.5	Both	8-10	2	2	1	2	3	5
7	PhD	0.6	Professional	1-2	5	5	5	5	4	3
8	PhD	12	Professional	10+	5	5	4	4	4	4
9	BSc	1	Academic	1-2	3	3	3	2	3	2
10	PhD	4.5	Professional	5-7	3	4	2	2	5	3
11	MSc	10	Academic	3-4	3	4	3	3	4	4
12	PhD	15	Professional	10+	5	4	3	4	5	5
13	PhD	10	Academic	5-7	4	3	2	3	3	3
14	MSc	8	Academic	10+	4	4	4	4	4	4
Level of Familiarity : too much = 5; a lot = 4; intermediate = 3; little = 2; very little = 1;										

Appendix B

The answers of the participants in the experiment to the first questionnaire are shown in Table 17.

Table 17: The answers to the first questionnaire

No.	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16
1	4	5	5	5	4	5	4	5	5	5	5	5	5	5	4	4
2	5	4	4	5	2	5	5	4	5	5	5	4	4	3	5	3
3	5	5	5	4	4	4	3	4	5	3	5	5	5	4	4	5
4	5	4	3	5	4	3	5	4	4	4	3	5	4	4	3	3
5	5	5	5	5	5	5	5	5	4	4	4	4	4	3	4	4
6	4	5	5	3	5	5	5	5	5	5	5	5	5	5	4	5
7	5	4	4	4	4	4	4	4	5	5	4	5	4	4	4	5
8	4	4	3	4	4	4	4	4	4	4	4	4	3	4	4	3
9	5	3	4	4	4	3	3	4	3	4	3	4	4	3	2	3
10	5	5	5	5	5	5	5	4	5	5	5	4	3	5	5	5
11	5	5	4	5	5	5	5	5	5	5	5	5	5	5	5	5
12	5	5	5	4	5	3	4	5	4	4	3	4	4	5	3	4
13	4	5	5	5	4	5	5	4	4	3	5	4	5	3	5	5
14	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
Average	4.71	4.57	4.43	4.5	4.29	4.36	4.43	4.43	4.5	4.36	4.36	4.5	4.29	4.14	4.07	4.21
Min	4	3	3	3	2	3	3	4	3	3	3	4	3	3	2	3
Max	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
SD[1]	0.47	0.65	0.76	0.65	0.83	0.84	0.76	0.51	0.65	0.74	0.84	0.52	0.73	0.86	0.92	0.89

[1] Standard Deviation (SD)

Appendix C

The answers of the participants in the experiment to the second questionnaire are shown in Table 18.

Table 18: The answers to the second questionnaire

No.	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
1	4	4	4	4	4	4	4	3	4	4	4
2	3	3	5	5	5	2	4	4	4	3	2
3	3	3	4	5	5	3	3	4	5	5	5
4	3	3	5	4	4	3	4	4	4	3	4
5	4	4	4	4	4	3	3	3	4	3	4
6	5	5	5	5	5	4	5	5	3	5	5
7	5	4	4	4	4	3	4	3	4	4	4
8	3	3	4	4	4	4	4	4	4	3	4
9	4	4	4	4	3	2	2	3	3	3	4
10	5	4	5	5	5	4	4	4	5	5	4
11	4	4	5	5	5	4	4	4	5	4	4
12	5	4	3	5	4	5	5	4	4	4	3
13	5	4	4	5	4	4	4	3	4	4	4
14	5	5	5	5	5	5	5	5	5	5	5
Average	4.14	3.86	4.36	4.57	4.36	3.57	3.93	3.79	4.14	3.93	4
Min	3	3	3	4	3	2	2	3	3	3	2
Max	5	5	5	5	5	5	5	5	5	5	5
SD	0.86	0.66	0.63	0.51	0.63	0.94	0.83	0.7	0.66	0.83	0.78

References

Akiki, P. A., & Maalouf, H. W. (2021). CHECKSUM: tracking changes and measuring contributions in cooperative systems modeling. *Software and Systems Modeling*, *20*(4), 1079–1122. https://doi.org/10.1007/s10270-020-00840-3

Alam, O., Corley, J., Masson, C., & Syriani, E. (2018). Challenges for reuse in collaborative modeling environments. *MODELS Workshops*, 277–283.

Asadi, M., & Ramsin, R. (2008). MDA-Based Methodologies: An Analytical Survey. In *Model Driven Architecture – Foundations and Applications* (pp. 419–431). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-69100-6_30

Bagherzadeh, M., Jahed, K., Combemale, B., & Dingel, J. (2021). Live modeling in the context of state machine models and code generation. *Software and Systems Modeling*, *20*(3), 795–819. https://doi.org/10.1007/s10270-020-00829-y

Basciani, F., Di Rocco, J., Di Ruscio, D., Iovino, L., & Pierantonio, A. (2016). Automated Clustering of Metamodel Repositories. In S. Nurcan, P. Soffer, M. Bajec, & J. Eder (Eds.), *Advanced Information Systems Engineering* (pp. 342–358). Springer International Publishing.

Basciani, F., Rocco, J., Di Ruscio, D., Di Salle, A., Iovino, L., & Pierantonio, A. (2014). MDEForge: An extensible Web-based modeling platform. In *CEUR Workshop Proceedings* (Vol. 1242).

Bruel, J.-M., Combemale, B., Guerra, E., Jézéquel, J.-M., Kienzle, J., de Lara, J., Mussbacher, G., Syriani, E., & Vangheluwe, H. (2020). Comparing and classifying model transformation reuse approaches across metamodels. *Software and Systems Modeling*, *19*(2), 441–465. https://doi.org/10.1007/s10270-019-00762-9

Bruneliere, H., de Kerchove, F. M., Daniel, G., Madani, S., Kolovos, D., & Cabot, J. (2020). Scalable model views over heterogeneous modeling technologies and resources. *Software and Systems Modeling*, *19*(4), 827–851. https://doi.org/10.1007/s10270-020-00794-6

Bucchiarone, A., Cabot, J., Paige, R. F., & Pierantonio, A. (2020). Grand challenges in model-driven engineering: an analysis of the state of the research. *Software and Systems Modeling*, *19*(1), 5–13. https://doi.org/10.1007/s10270-019-00773-6

Chavarriaga, J., Hoyos, H., & Gómez, P. (2014, September). *Solving the FIXML Case Study Using Epsilon and Java*. https://ceur-ws.org/Vol-1305/ttc14.pdf#page=93

Combemale, B., Deantoni, J., Baudry, B., France, R., Jézéquel, J.-M., & Gray, J. (2014). Globalizing Modeling Languages. *Computer*, *47*, 68–71. https://doi.org/10.1109/MC.2014.147

Corley, J., & Syriani, E. (2014). A Cloud Architecture for an Extensible Multi-Paradigm Modeling Environment. *PSRC@ MoDELS*, 6–10.

Cuadrado, J. S., Guerra, E., & Lara, J. de. (2014). A Component Model for Model Transformations. *IEEE Transactions on Software Engineering*, *40*(11), 1042–1060. https://doi.org/10.1109/TSE.2014.2339852

Daniel, G., Sunyé, G., & Cabot, J. (2019). Advanced prefetching and caching of models with PrefetchML. *Software & Systems Modeling*, *18*(3), 1773–1794. https://doi.org/10.1007/s10270-018-0671-8

David, I., Latifaj, M., Pietron, J., Zhang, W., Ciccozzi, F., Malavolta, I., Raschke, A., Steghöfer, J.-P., & Hebig, R. (2023). Blended modeling in commercial and open-source model-driven software engineering tools: A systematic study. *Software and Systems Modeling*, *22*(1), 415–447. https://doi.org/10.1007/s10270-022-01010-3

Di Rocco, J., Di Ruscio, D., Pierantonio, A., Cuadrado, J. S., de Lara, J., & Guerra, E. (2016). Using ATL Transformation Services in the MDEForge Collaborative Modeling Platform. In P. Van Gorp & G. Engels (Eds.), *Theory and Practice of Model Transformations* (pp. 70–78). Springer International Publishing.

Diskin, Z., Wider, A., Gholizadeh, H., & Czarnecki, K. (2014). *Towards a Rational Taxonomy for Increasingly Symmetric Model Synchronization* (pp. 57–73). https://doi.org/10.1007/978-3-319-08789-4_5

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). *Microservices: Yesterday, Today, and Tomorrow BT - Present and Ulterior Software Engineering* (M. Mazzara & B. Meyer, Eds.; pp. 195–216). Springer International Publishing. https://doi.org/10.1007/978-3-319-67425-4_12

EMF.cloud - Evolve your modeling tools to the web! (n.d.). Retrieved October 30, 2023, from https://eclipse.dev/emfcloud/

Eugene Syriani, Hans Vangheluwe, Raphael Mannadiar, Conner Hansen, Simon Van Mierlo, & Huseyin Ergin. (n.d.). *AToMPM: A Web-based Modeling Environment*. Retrieved October 30, 2023, from https://ceur-ws.org/Vol-1115/demo4.pdf

Franzago, M., Ruscio, D. D., Malavolta, I., & Muccini, H. (2018). Collaborative Model-Driven Software Engineering: A Classification Framework and a Research Map. *IEEE Transactions on Software Engineering*, *44*(12), 1146–1175. https://doi.org/10.1109/TSE.2017.2755039

Gómez, A., Mendialdua, X., Barmpis, K., Bergmann, G., Cabot, J., de Carlos, X., Debreceni, C., Garmendia, A., Kolovos, D. S., & de Lara, J. (2020). Scalable modeling technologies in the wild: an experience report on wind turbines control applications development. *Software and Systems Modeling*, *19*(5), 1229–1261. https://doi.org/10.1007/s10270-020-00776-8

Hailpern, B., & Tarr, P. (2006). Model-driven development: The good, the bad, and the ugly. *IBM Systems Journal*, *45*(3), 451–461. https://doi.org/10.1147/sj.453.0451

Heinrich, R., van Hoorn, A., Knoche, H., Li, F., Lwakatare, L. E., Pahl, C., Schulte, S., & Wettinger, J. (2017). *Performance Engineering for Microservices: Research Challenges and Directions*. https://doi.org/10.1145/3053600.3053653

Holmes, T., Zdun, U., & Dustdar, S. (2012). Automating the Management and Versioning of Service Models at Runtime to Support Service Monitoring. *2012 IEEE 16th International Enterprise Distributed Object Computing Conference*, 211–218. https://doi.org/10.1109/EDOC.2012.32

Ivanchikj, A., Serbout, S., & Pautasso, C. (2022). Live process modeling with the BPMN Sketch Miner. *Software and Systems Modeling*, *21*(5), 1877–1906. https://doi.org/10.1007/s10270-022-01009-w

Jahed, K., Bagherzadeh, M., & Dingel, J. (2021). On the benefits of file-level modularity for EMF models. *Software and Systems Modeling*, *20*(1), 267–286. https://doi.org/10.1007/s10270-020-00804-7

Kolovos, D., Rose, L. M., Matragkas, N., Paige, R. F., Guerra, E., Sánchez Cuadrado, J., de Lara, J., Ráth, I., Varró, D., Tisi, M., & Cabot, J. (2013). A research roadmap towards achieving scalability in model driven engineering. In *Workshop on Scalability in Model Driven Engineering 2013*. ACM. https://doi.org/10.1145/2487766.2487768

Kolovos, D. S., Rose, L. M., Paige, R. F., Guerra, E., Sánchez Cuadrado, J., Lara, J., Ráth, I., Varro, D., Sunyé, G., & Tisi, M. (2015). MONDO: Scalable Modelling and model management on the cloud. *CEUR Workshop Proceedings*, *1400*, 44–53.

Kusel, A., Schönböck, J., Wimmer, M., Kappel, G., Retschitzegger, W., & Schwinger, W. (2015). Reuse in model-to-model transformation languages: are we there yet? *Software & Systems Modeling*, *14*(2), 537–572. https://doi.org/10.1007/s10270-013-0343-7

Lara, J. De, & Guerra, E. (2017). A Posteriori Typing for Model-Driven Engineering. *ACM Transactions on Software Engineering and Methodology*, *25*(4), 1–60. https://doi.org/10.1145/3063384

Lara, J. De, Guerra, E., & Cuadrado, J. S. (2014). When and How to Use Multilevel Modelling. *ACM Transactions on Software Engineering and Methodology*, *24*(2), 1–46. https://doi.org/10.1145/2685615

Legros, E., Amelunxen, C., Klar, F., & Schürr, A. (2009). Generic and reflective graph transformations for checking and enforcement of modeling guidelines. *Journal of Visual Languages & Computing*, *20*(4), 252–268. https://doi.org/https://doi.org/10.1016/j.jvlc.2009.04.005

López, J. A. H., & Cuadrado, J. S. (2022). An efficient and scalable search engine for models. *Software and Systems Modeling*, *21*(5), 1715–1737. https://doi.org/10.1007/s10270-021-00960-4

Maróti, M., Kecskés, T., Kereskényi, R., Broll, B., Völgyesi, P., Jurácz, L., Levendovszky, T., & Lédeczi, Á. (2014). Next generation (meta) modeling: web-and cloud-based collaborative tool infrastructure. *MPM@ MoDELS*, *1237*, 41–60.

Mkaouar, H., Blouin, D., & Borde, E. (2023). A benchmark of incremental model transformation tools based on an industrial case study with AADL. *Software and Systems Modeling*, *22*(1), 175–201. https://doi.org/10.1007/s10270-022-00989-z

Ogunyomi, B., Rose, L. M., & Kolovos, D. S. (2019). Incremental execution of model-to-text transformations using property access traces. *Software & Systems Modeling*, *18*(1), 367–383. https://doi.org/10.1007/s10270-018-0666-5

Panahandeh, M., Hamdaqa, M., Zamani, B., & Hamou-Lhadj, A. (2021). MUPPIT: a method for using proper patterns in model transformations. *Software and Systems Modeling*, *20*(5), 1491–1523. https://doi.org/10.1007/s10270-020-00853-y

Popoola, S., Carver, J. C., & Gray, J. (2017). Modeling as a Service: A Survey of Existing Tools. *MODELS*.

Rademacher, F., Sachweh, S., & Zündorf, A. (2017). Differences between Model-Driven Development of Service-Oriented and Microservice Architecture. *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, 38–45. https://doi.org/10.1109/ICSAW.2017.32

Richardson, C. (2018). *Microservices patterns : with examples in Java*.

Robson, C., & McCartan, K. (2017). *Real World Research*, *4th Edition*.

Rocco, J. Di, Ruscio, D. Di, Iovino, L., & Pierantonio, A. (2015). Collaborative Repositories in Model-Driven Engineering [Software Technology]. *IEEE Software*, *32*(3), 28–34. https://doi.org/10.1109/MS.2015.61

Rodrigues da Silva, A. (2015). Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, *43*, 139–155. https://doi.org/https://doi.org/10.1016/j.cl.2015.06.001

Shamsujjoha, Md., Grundy, J., Li, L., Khalajzadeh, H., & Lu, Q. (2021). Developing Mobile Applications Via Model Driven Development: A Systematic Literature Review. *Information and Software Technology*, *140*, 106693. https://doi.org/10.1016/j.infsof.2021.106693

Shin, S.-S. (2019). Empirical study on the effectiveness and efficiency of model-driven architecture techniques. *Software & Systems Modeling*, *18*(5), 3083–3096. https://doi.org/10.1007/s10270-018-00711-y

Steel, J., & Jézéquel, J.-M. (2007). On model typing. *Software & Systems Modeling*, *6*(4), 401–413. https://doi.org/10.1007/s10270-006-0036-6

Tröls, M. A., Marchezan, L., Mashkoor, A., & Egyed, A. (2022). Instant and global consistency checking during collaborative engineering. *Software and Systems Modeling*, *21*(6), 2489–2515. https://doi.org/10.1007/s10270-022-00984-4

Vahdati, A., & Ramsin, R. (2022). *Modeling and Model Transformation as a Service: Towards an Agile Approach to Model-Driven Development* (pp. 116–135). https://doi.org/10.1007/978-3-030-94238-0_7

Vahdati, A., & Ramsin, R. (2023). Microservice-based model-driven architecture for implementing modeling and model transformation as a service (Appendices), Mendeley Data, http://dx.doi.org/10.17632/p9wtnmcj7j

Wagelaar, D., Tisi, M., Cabot, J., & Jouault, F. (2011). Towards a General Composition Semantics for Rule-Based Model Transformation. In J. Whittle, T. Clark, & T. Kühne (Eds.), *Model Driven Engineering Languages and Systems* (pp. 623–637). Springer Berlin Heidelberg.

Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W., Schoenboeck, J., & Schwinger, W. (2010). Surviving the Heterogeneity Jungle with Composite Mapping Operators. In L. Tratt & M. Gogolla (Eds.), *Theory and Practice of Model Transformations* (pp. 260–275). Springer Berlin Heidelberg.

Wimmer, M., Kusel, A., Retschitzegger, W., Schönböck, J., Schwinger, W., Sánchez Cuadrado, J., Guerra, E., & Lara, J. (2012). Reusing Model Transformations across Heterogeneous Metamodels. *Electronic Communications of the EASST*, *50*. https://doi.org/10.14279/tuj.eceasst.50.722.795