Model Transformation Chains as Strategy for Software Development Projects

Hector Florez

Universidad Distrital Francisco Jose de Caldas, Bogota, Colombia

Abstract

Currently, there are several enterprises around the world dedicated to the same business, e.g., banking, academy, trading, etc. All of those enterprises require specific information systems; however, information systems in the same business have several similar or equal features. Some applications are based on frameworks; however, most features introduce a lot of source code, impacting the performance of an application, especially when the application is web-based. In this paper, three case studies based on Model Transformation Chains are presented. These case studies are focused on the generation of final applications based on an input conceptual model following the most important concepts of Model-Driven Engineering.

Keywords

Model Driven Engineering (MDE), Model Transformation Chain (MTC), Eclipse Modeling Framework (EMF), Conceptual models

1. Introduction

Software products need to provide the functionalities specified in the business functional requirements as well as to support flexibility to change. Then, software products demand an architecture to offer this characteristic [1].

Furthermore, in Model-Driven Engineering, a metamodel is used to make an abstraction of a specific domain, while a model, which conforms to a metamodel, is used to make a representation of a specific case in the modeled domain, where the model has to follow the metamodel's structure and constraints [2, 3]. Moreover, a model transformation allows converting a model that conforms to a source metamodel into a new model that conforms to a target metamodel [4]. A transformation is used to add valuable components to a source model. In several cases, the final result after running a model transformation is the source code of a software project [5].

This paper presents three case studies where the Model Transformation Chains (MTC) allow creating a final project based on an input conceptual model. The first case study presents an approach to create mobile Android applications with peripheral configuration. The second case study presents an approach to generate web applications based on an input conceptual model written in XML. Finally, the third case study presents the construction of an ArchiMate editor to analyze enterprise models.

The remainder of the paper is as follows. Section 2 presents the main concepts related to the case studies presented in the paper.

2. Background

2.1. Model Driven Engineering

Model-Driven Engineering (MDE) is an approach that offers the required concepts to understand the elements related to metamodels and models. Then, MDE uses models and metamodels to support the design, construction, deployment, maintenance, and upgrades of a system.

IWSAI 2026: 1st International Workshop on Systems, Applications, Artificial Intelligence and Innovation, September 16–18, 2026, Piura, Peru

*Corresponding author.

haflorezf@udistrital.edu.co (H. Florez)

0000-0002-5339-4459 (H. Florez)

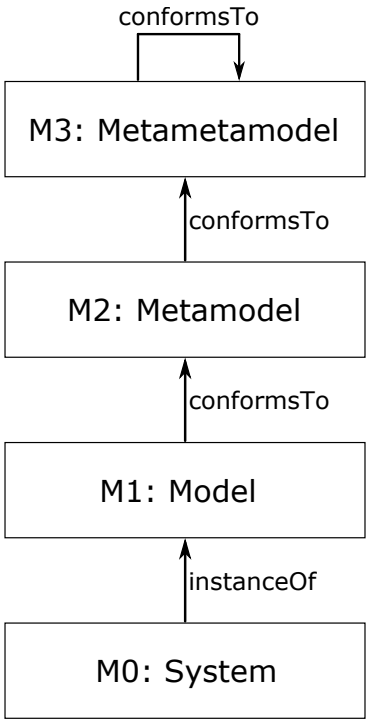

Figure 1: Four Levels Architecture.

MDE is an approach to address technologies regarding Domain Specific Modeling Languages (DSML) as well as transformation and generation engines [6]. In software engineering, a model refers to an artifact developed in a modeling language, which describes a system through different types of diagrams. One model requires three key features [7, 8]:

- Mapping, indicating that a model is based on an original concept.
- Reduction, indicating that the model reflects a specific selected part of the original concept.
- Pragmatic concept, indicating that the model needs to be used taking the specific purpose of the original concept.

In the context of MDE, every model needs to meet the following characteristics [9, 10]:

- Abstraction. It means that a model must be a simplification of the modeled system.
- Understandability. It means that a model must be intuitive.
- Accuracy. It means that a model must provide a correct representation of the modeled system.
- Predictiveness. It means that a model must predict the most important characteristics of the modeled system.
- Inexpensiveness. It means that the construction of a model must be significantly cheaper than the modeled system.

The Object Management Group (OMG)[1] proposed a four-layered architecture presented in Figure 1. The OMG called these layers `M0`, `M1`, `M2`, and `M3`. Layer `M0` corresponds to the instances of the system under study. Layer `M1` includes the model of the system. The layer `M2` includes metamodels that define the concepts and relations of the domain that meets the system under study. Finally, layer `M3` defines a meta-metamodel that presents a definition of the elements that can be created in the metamodel.

This architecture also defines relations between layers. One instance of layer `M0` is an *instanceOf* a model placed in the layer `M1`, which `conformsTo` a metamodel placed in the layer `M2`, which *conformsTo* a meta-metamodel placed in the layer `M3`. However, the meta-metamodel *conformsTo* itself [5].

[1] http://www.omg.org/

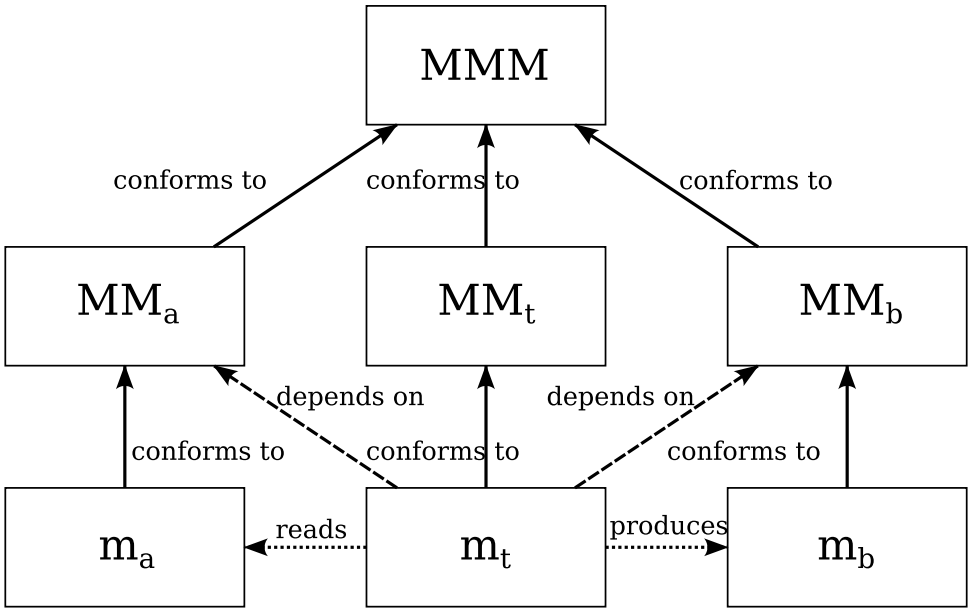

Figure 2: Model Transformation.

2.2. Model Transformations

Model transformations are considered an important application of MDE when using models as artifacts in the process of code generation. A transformation is defined as the change of a source model, which conforms to a source metamodel, into a target model, which conforms to a target metamodel [11]. Figure 2 depicts the transformation. To make this transformation, it is necessary to use a transformation language. In a model transformation, the transformation language is defined by a metamodel (MM_t). Then, the source metamodel (MM_a), target metamodel (MM_b), and transformation language metamodel (MM_t) must conform to one meta-metamodel (MMM)(e.g., ECORE). In this structure, there is a transformation specification, which is source code created using the transformation language and is considered a model (m_t) that conforms to the transformation language. This transformation specification depends on the elements defined in the source and target metamodels. Finally, the transformation execution takes the source model (m_t) and generates the target model (m_t).

2.3. Domain Specific Modeling Language (DSML)

DSMLs formalize the structure of particular domains, and they are described using metamodels that contain the domain concepts and rules. These metamodels are the basis for creating models of systems that belong to the described domain. The DSML metamodel conforms to the metametamodel provided by the technology used to create the DSMLs.

DSMLs are created to build domain models with an intended purpose. For instance, some DSMLs allow applying domain-specific analyses to models.

3. Case Studies

This section presents three case studies that use Model Driven Engineering and Model Transformation Chains to generate projects with different purposes.

3.1. Android Applications

This project is focused on the configuration of mobile device peripherals for the Android operating system. To achieve this, the model transformation chain presented in Figure 3 was implemented. This

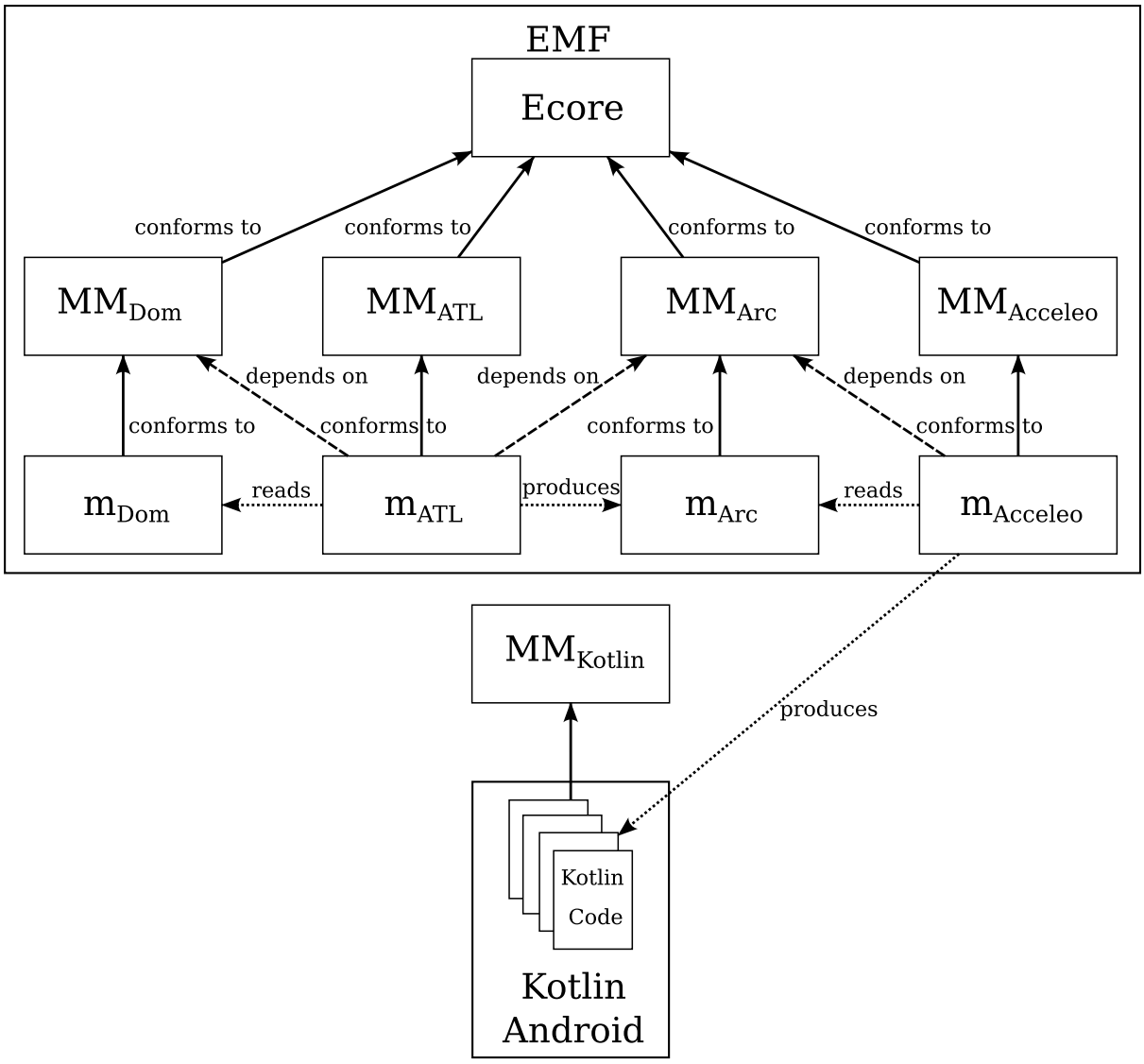

Figure 3: Model Transformation Chain for Android Applications.

project was developed using the Eclipse Modeling Framework (EMF)[2]; thus, the selected metametamodel is $Ecore$ because it is used in EMF.

The first transformation starts with a domain metamodel MM_{Dom}, which conforms to $Ecore$ and contains all required elements related to the specific domain, which in this case corresponds to the development for mobile devices, specifically for peripheral configurations. In addition, the Atlas Transformation Language (ATL) metamodel MM_{ATL}, which conforms to $Ecore$ contains the elements for developing model-to-model transformations using the ATL language. Furthermore, an architecture metamodel MM_{Arc}, which conforms to $Ecore$ contains all the elements of the MM_{Dom}, adding elements related to Android architectural concepts, object-oriented paradigm concepts in the Kotlin programming language, MDA concepts, and clean architecture concepts.

The second transformation starts with the MM_{Arc}. It also includes the Acceleo Metamodel $M_{Acceleo}$, which conforms to $Ecore$ and and contains the elements for developing model-to-text transformations using the Acceleo language. In addition, the Kotlin Metamodel MM_{Kotlin} contains the elements that reference the concepts of the Kotlin programming language.

Then, a domain model m_{Dom}, which conforms to MM_{Dom}, is transformed to an architecture model m_{Arc}, which conforms to MM_{Arc}, based on the specifications created in the ATL model m_{ATL}, which

[2]https://eclipse.dev/emf/

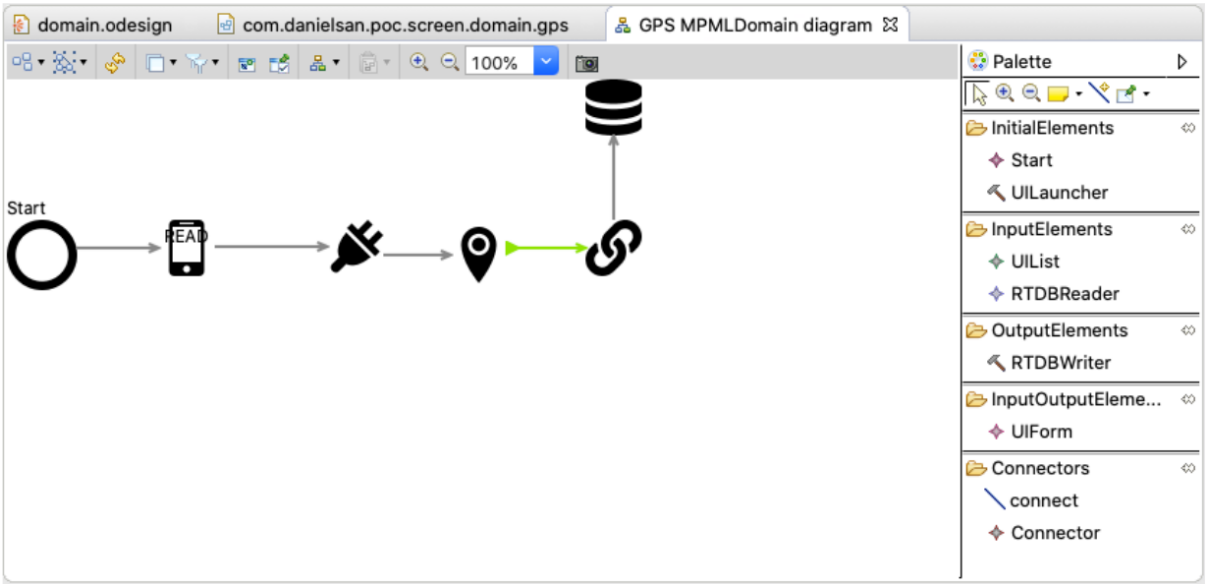

Figure 4: Domain model created with a DSML that conforms to the MPML.

conforms to MM_{Arc}. Finally, the Acceleo model $m_{Acceleo}$, which conforms to $MM_{Acceleo}$ transforms the m_{Arc} to source code that conforms to MM_{Kotlin} [5, 12].

In this project, the MM_{Dom} is called *Mobile Peripheral Modeling Language (MPML)*, which not only allows abstraction of the business logic but also allows creating a DSML that allows creating the m_{Dom} supported by a graphical interface that includes icons of the business elements and a flowchart to represent the interactions between the modeled elements. Figure 4 presents a screenshot of an m_{Dom} created using the DSML that conforms to $MPML$. The palette on the right contains all elements that conform to the business logic entities modeled in the $MPML$.

3.2. Web Applications

This project, called *DevPHP*, is focused on a model transformation to generate source code for transactional web projects in the PHP language based on a conceptual model written in XML. Figure 5 presents the flow to run the transformation. First, the modeler is the person who creates the XML model. Later,

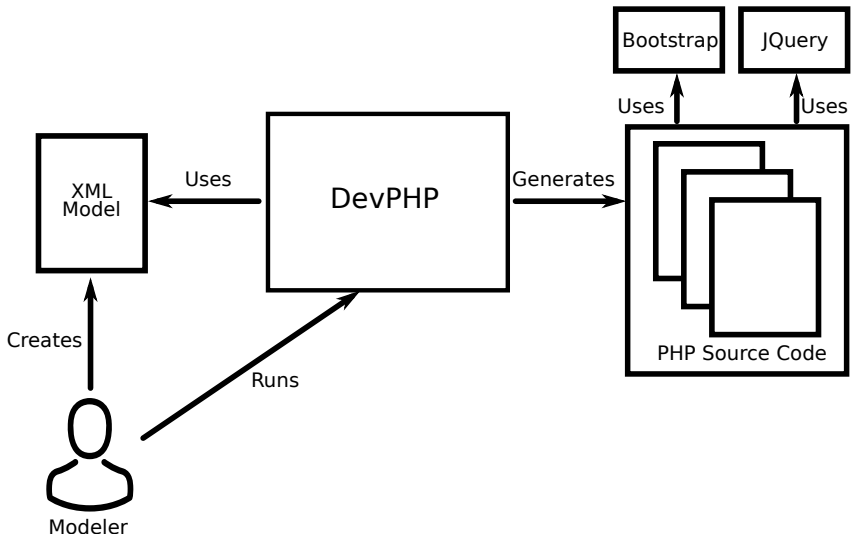

Figure 5: Transformation flow.

the modeler runs *DevPHP*, which uses the XML model to generate PHP source files that compose a PHP project. Moreover, additional folders are created to support the generated project. Those folders are: *css* for *Bootstrap* cascade style sheet files, *js* for Bootstrap, textit jQuery, and Validator JavaScript files, img for images, and uml for creating the UML model to be used in UML Designer by Obeo [11].

The XML model follows a specific structure based on the following XML elements: a) `model`, which is the main XML element; b) `entity`, which is the element that allows modelers to create the concepts involved in the project; c) `attribute`, which is the element that allows including attributes in entities and must be included in the context of the element entity; d) `relation`, which is the element that must be included in the context of the element entity and serves to include one relation to another entity; and e) `service`, which is the element that must be included in the context of the element entity only when the attribute actor in the element entity has the value true and serves to define which entities can be controlled by a desired actor.

Listing 1 presents a fragment of the XML model for a project that we called *RIS (Research Information System)*. *RIS* is intended to manage research information of a research group. Thus, *RIS* has the entities: *Researcher Role*, *Researcher*, *Book*, *Book Chapter*, *Paper*, *Software*, and *Project* [11].

Listing 1: Fragment of XML Model

```
<model name="Research Information System" acronym="RIS" description="Research Information System allows managing the research information of a research group" language="en" >
  <entity name="ResearcherRole" delete="true" >
    <attribute name="name_en" type="string" mandatory="true" />
    <attribute name="name_es" type="string" mandatory="true" />
    <relation cardinality="*" entity="Researcher" />
  </entity>
  <entity name="Researcher" actor="true" >
    <attribute name="name" type="string" mandatory="true" />
    <attribute name="lastName" type="string" mandatory="true" />
    <attribute name="email" type="email" />
    <attribute name="password" type="password" />
    <attribute name="picture" type="string" image="true" visible="false" />
    <attribute name="isi" type="string" length="200" url="true" visible="false" />
    <attribute name="scopus" type="string" length="200" url="true" visible="false" />
    <relation cardinality="1" entity="ResearcherRole" />
    <relation cardinality="*" entity="PaperResearcher" />
    <service entity="Researcher" create="false" get="true" edit="false" delete="false" />
    <service entity="Paper" create="true" get="true" edit="true" delete="false" />
  </entity>
  <entity name="Paper" delete="true" >
    <attribute name="title" type="string" mandatory="true" length="200" deploy="true" />
    <attribute name="authors" type="string" mandatory="true" length="200" />
    <attribute name="journal" type="string" length="100" />
    <attribute name="issn" type="string" nowrap="true" />
    <attribute name="volume" type="string" visible="false" />
    <attribute name="pages" type="string" visible="false" />
    <attribute name="year" type="int" />
    <attribute name="doi" type="string" length="200" url="true" />
    <relation cardinality="*" entity="PaperResearcher" />
  </entity>
  <entity name="PaperResearcher" delete="true" >
    <relation cardinality="1" entity="Paper" />
    <relation cardinality="1" entity="Researcher" />
  </entity>
</model>
```

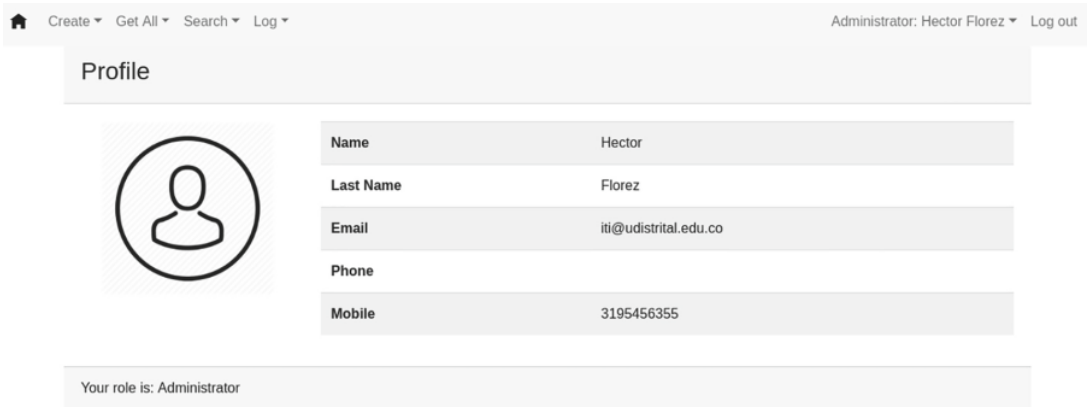

Figure 6: Session page of the administrator.

Figure 6 presents the index page of the generated PHP project RIS. On this page, actors can log in through their email and password using the card located on the right. It also offers the option to recover the password. This option sends a new random password to the email registered by the user who is requesting the password recovery. In the card located in the center, it shows the project description that is included in the XML conceptual model [11].

3.3. ArchiMate Modeling Language

This project, called *iArchiMate*, is a DSML for modeling and enterprise models focused on the ArchiMate modeling language. In particular, *iArchiMate* serves to create imperfect ArchiMate models in a flexible way by creating drafts of models as well as to analyze such enterprise models. Imperfect ArchiMate models refer to models that include imperfect information. The tool's core is a graphical editor based on EMF [8].

Figure 7 presents the iArchiMate metamodel, which allows creating ArchiMate models that conform to the ArchiMate modeling language, while Figure 8 presents a fragment of a layered view for an *iArchiMate* model.

iArchiMate handles *linguistic conformance* using EMF's validation engine, and *ontological conformance* using iArchiMate's validation engine [13], which is based on Epsilon Validation Language (EVL). Since *iArchiMate* was created to build imperfect models, the *iArchiMate* metamodel has been extended to

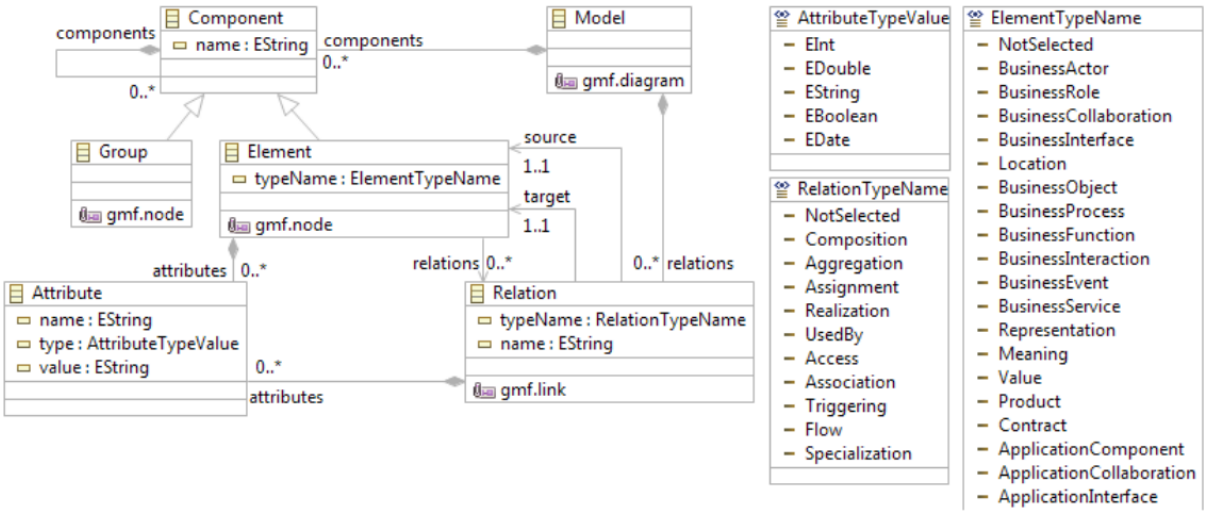

Figure 7: *iArchiMate* metamodel.

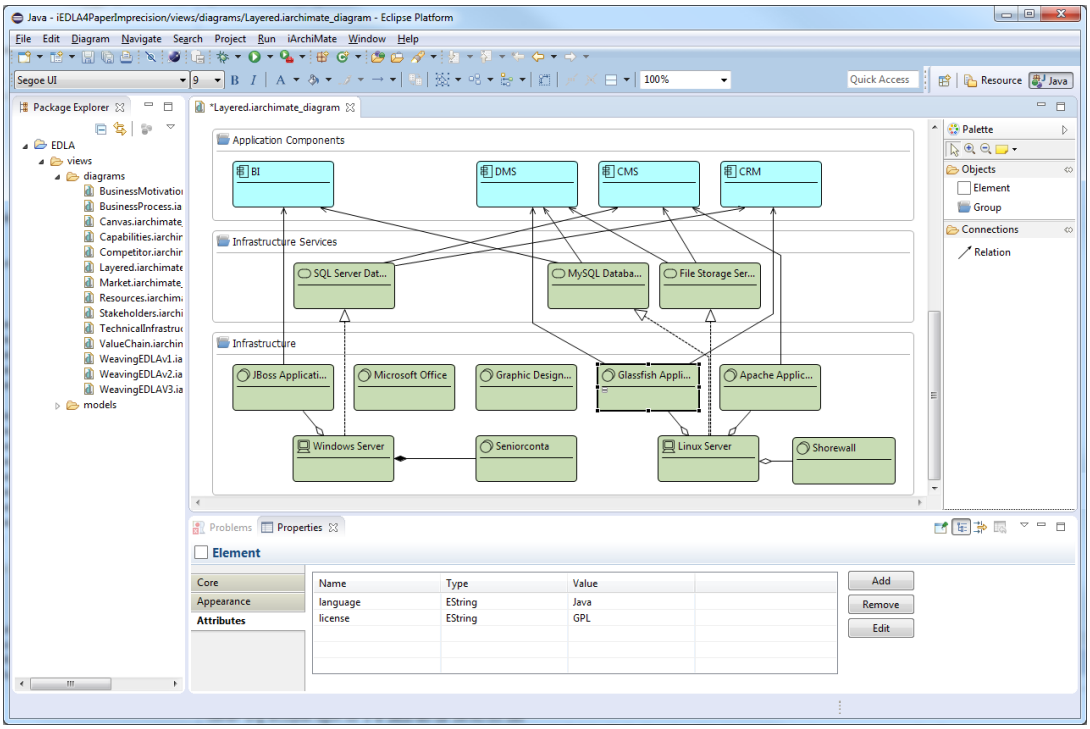

Figure 8: Fragment of a layered view for an *iArchiMate* model.

include a decision trace that specifies the reasons why the model includes certain imperfect information. Figure 9 presents the *iArchiMate* extension, which includes the necessary concepts to represent the decision trace entities.

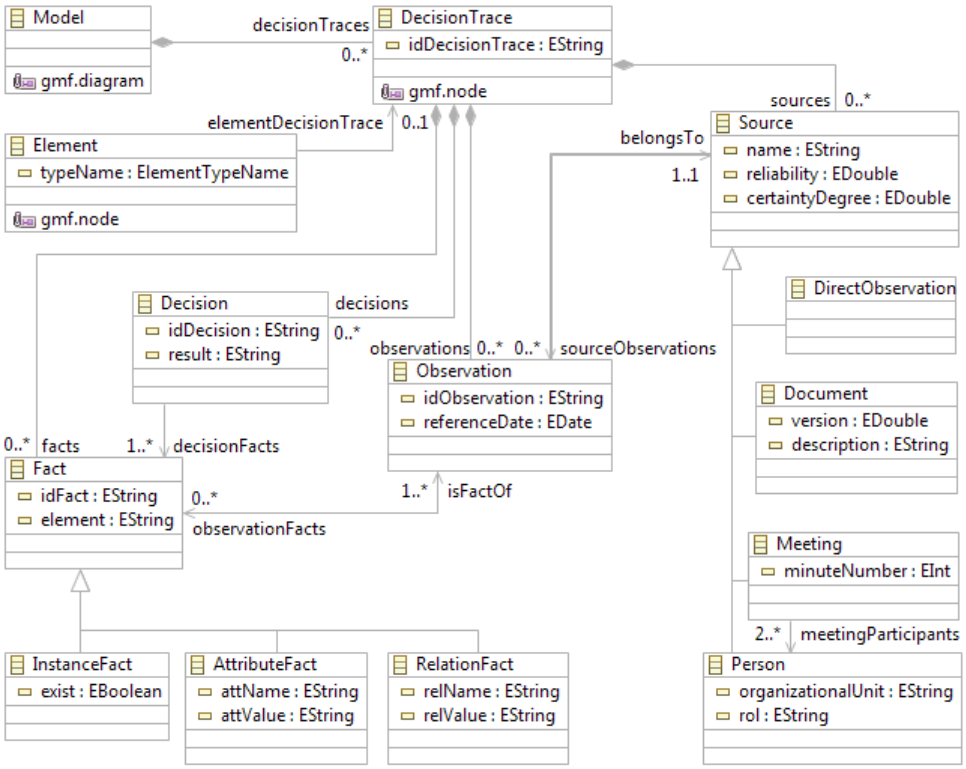

Figure 9: Extended *iArchiMate* metamodel.

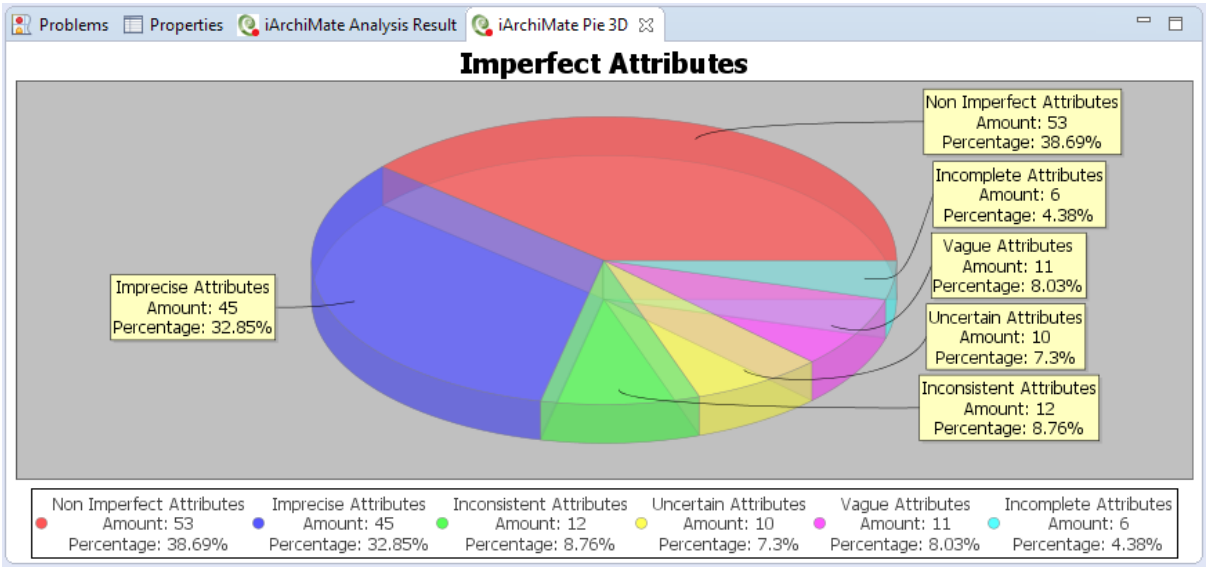

Figure 10: Extended *iArchiMate* metamodel.

Finally, when running analyses, *iArchiMate* provides various views for delivering the analysis results to support decision-making processes in enterprises. Figure 10 presents the results of running an analysis to measure the imperfection level in the enterprise model.

4. Conclusions

Model-driven engineering approaches provide the required concepts to create model transformation chains to automatically generate software projects based on an input conceptual model. In addition, metamodels describe not only the structure of the domain but may also describe the required elements to deploy a domain-specific modeling language, enabling modelers to create their conceptual models using graphical tools. Moreover, the metamodels can describe elements to enrich the models with additional features such as model analysis.

Declaration on Generative AI

The author has not employed any Generative AI tools.

References

[1] D. Sanchez, O. Mendez, H. Florez, An approach of a framework to create web applications, in: Lecture Notes in Computer Science, Springer, 2018, pp. 341–352. URL: http://link.springer.com/10.1007/978-3-319-95171-3_27. doi:10.1007/978-3-319-95171-3_27.

[2] H. Florez, M. Sánchez, J. Villalobos, G. Vega, Coevolution assistance for enterprise architecture models, in: Proceedings of the 6th International Workshop on Models and Evolution, ACM, 2012, pp. 27–32. URL: https://dl.acm.org/doi/10.1145/2523599.2523605. doi:10.1145/2523599.2523605.

[3] P. Gómez, M. Sánchez, H. Florez, J. Villalobos, An approach to the co-creation of models and metamodels in enterprise architecture projects, Journal of Object Technology 13 (2014) 1–29. URL: https://www.jot.fm/contents/issue_2014_07/article2.html. doi:10.5381/jot.2014.13.3.a2.

[4] H. Florez, M. Leon, Model driven engineering approach to configure software reusable components, in: Communications in Computer and Information Science, Springer, 2018, pp. 352–363. URL: http://link.springer.com/10.1007/978-3-030-01535-0_26. doi:10.1007/978-3-030-01535-0_26.

[5] D. Sanchez, A. E. Rojas, H. Florez, Towards a clean architecture for android apps using model transformations, IAENG International Journal of Computer Science 49 (2022) 1–9. URL: https://www.iaeng.org/IJCS/issues_v49/issue_1/IJCS_49_1_28.pdf.

[6] D. C. Schmidt, et al., Model-driven engineering, Computer-IEEE Computer Society- 39 (2006) 25.

[7] T. Kühne, Matters of (meta-) modeling, Software & Systems Modeling 5 (2006) 369–385.

[8] H. Florez, M. Sanchez, J. Villalobos, iarchimate: A tool for managing imperfection in enterprise models, in: 2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations, IEEE, 2014, pp. 201–210. URL: http://ieeexplore.ieee.org/document/6975362/. doi:10.1109/EDOCW.2014.38.

[9] B. Selic, The pragmatics of model-driven development, IEEE software 20 (2003) 19–25.

[10] P.-A. Muller, F. Fondement, B. Baudry, B. Combemale, Modeling modeling modeling, Software & Systems Modeling 11 (2012) 347–359.

[11] H. Florez, E. Garcia, D. Muñoz, Automatic code generation system for transactional web applications, in: Lecture Notes in Computer Science, Springer, 2019, pp. 436–451. URL: http://link.springer.com/10.1007/978-3-030-24308-1_36. doi:10.1007/978-3-030-24308-1_36.

[12] D. Sanchez, H. Florez, Model driven engineering approach to manage peripherals in mobile devices, in: Lecture Notes in Computer Science, Springer, 2018, pp. 353–364. URL: http://link.springer.com/10.1007/978-3-319-95171-3_28. doi:10.1007/978-3-319-95171-3_28.

[13] H. Florez, M. Sanchez, J. Villalobos, Analysis of imprecise enterprise models, in: Lecture Notes in Business Information Processing, Springer, 2016, pp. 349–364. URL: http://link.springer.com/10.1007/978-3-319-39429-9. doi:10.1007/978-3-319-39429-9.