Implementation of a Domain Metamodel for the Generation of UML Documentation through Model Transformation Chains

Jeyson Stith Arévalo Sandoval

Universidad Distrital Francisco José de Caldas, Bogotá, Colombia

Abstract

UML has become a language that allows modeling the components of a process for building software that provides a modeling language, the success of UML is its use in successful projects and the increase in the need for support tools and documentation for software. However, in spite of being a fundamental part, of the generation of the documentation of the code and components involved in the construction of software, the importance of documentation in a project is not given enough value. This paper proposes the theoretical bases that allow the generation of rules for the generation of a domain metamodel of UML diagrams. The rules are defined allowing a specification where ambiguities and the need to learn a specific programming language are avoided thanks to the use of a modeling framework. This is where model transformation chains are an alternative to propose a pattern that identifies an input model (domain-specific metamodel that are the diagrams that make up the UML documentation) and convert it into an expected output model that generates value to the documentation in a software project. For the development of the proposed metamodel, the stages of research and review of documentation of the EMF tool, analysis, and implementation of the general components and construction of the project were carried out.

Keywords

Metamodel, Model, UML, EMF

1. Introduction

The software life cycle includes a series of phases, called: definition, analysis, design, implementation, deployment, testing, and maintenance [1]. In the first four phases, sets of diagrams are created with levels of abstraction that decrease as the project progresses and gradually become the software code; this process is often referred to as refinement. Despite the differences in the levels of abstraction, the diagrams of the different phases must represent the same system, so that the concepts of the system domain that were identified in the definition phase must still be preserved in the implementation [2]. This means that there must be consistency between the diagrams of the different phases of the project.

UML has been widely used to model information systems and in recent years it is regaining prominence thanks to Model Driven Engineering since it is the preferred language for the most popular approaches in this topic [3, 4].

ICAIW 2022: Workshops at the 5th International Conference on Applied Informatics 2022, October 27–29, 2022, Arequipa, Peru

jeysonarevalo@hotmail.com (J. S. Arévalo Sandoval)

Eclipse is one of the best open-source development platforms and it is highly extensible through plugins, there are many projects around a large community of users and developers [5]. The Eclipse workbench is made up of workspace, wizards, editors, views, and perspectives. Eclipse Modeling Framework (EMF)[1] is the core of the Eclipse platform for model-driven development, it is the framework for the development of metamodels (abstract syntax) that allows automatically generating Java implementation classes for the elements of our metamodels.

For the generation of the specific domain metamodel of our case study, functionalities and EMF projects must be established, to then build the Ecore metamodel corresponding to the context that surrounds UML.

2. Related work

In the context of software construction, the documentation of projects has become very important, so much so that it is currently considered part of the applications. The generation of the documentation saves several features, from the methodology used for its implementation to the syntax, notation, and visualization with which it is represented. The importance of documentation lies especially in the fact that what is expressed with the diagrams and mechanisms used must be totally consistent with the totality of the project it represents, referring to any aspect involved in its development. This will involve the entire context of infrastructure, technology, methodology, framework, and architecture related to the represented system.

The purpose of Model Driven Engineering (MDE) is to allow the representation of reality through models [6, 7], which facilitates the implementation of information systems. However, as already mentioned, despite being a fundamental part, the generation of the documentation of the code and components involved in its construction is not given importance, that is where the model transformation chains are an alternative to take. control, due to the possibility they offer to propose a pattern that identifies an input model and converts it into an expected output model. So, it is possible to obtain different solutions not only for an application but also for the representation of the system, which in our case are UML diagrams.

Bézivin [8] states that modeling is essential for human activity because every action is preceded by the Construction (implicit or explicit) of a model, there are a lot of practical uses of models: statistical models, meteorological models, models biological models, ecological models, economic models, etc. Informatics can be primarily described as the science of building software models. If the use of the word in the engineering sciences model is recent, the idea itself dates to ages.

One important difference between the old modeling practices and modern MDE is that the new vision is not to use models only as simple documentation but as formal input/output for computer-based tools implementing precise operations. As a consequence model-engineering frameworks have progressively evolved towards solid proposals like the MDA defined by the OMG. We may clearly see the three levels of principles (general MDE principles as described in this paper), of standards (e.g. the OMG/MDA set of standards), and of tools (like the EMF, Eclipse Modeling Framework, or the Visual Studio Team system).

[1]www.eclipse.org/modeling/emf/

Progressively we come to understand much better the possibilities and limits of this new way of considering information systems. The notion of a metamodel is strongly related to the notion of ontology. A metamodel is a formal specification of abstraction, usually consensual and normative. From a given system we can extract a particular model with the help of a specific metamodel. A metamodel acts as a precisely defined filter expressed in a given formalism [9].

Ludewig [10] mentions that the term "model" is derived from the Latin word modulus, which means measure, ruler, and model. Models are intended to assist in the development of artifacts to provide tools for the construction of components that represent facts.

Diagrammatic representation allows the reuse of potential applications as part of a larger system. For example, if the construction of user authentication and registration system is required, the schematization of this requirement will be equally valid for a banking system, a health care system, or an academic system.

Florez [11] explains the advantage of using model transformation chains stating different contexts related to the development of software products, it is possible to see that, in general, most of the functional requirements that represent around the same context, respond to the same set of characteristics. Generally, software developers reuse code generated in previous projects that solves certain requirements to improve the development time of a particular product. This code reuse is an immediate solution that does not necessarily solve the problem in a new project due to the specific situations in the new project. These situations in almost all cases have differences from the application made in previous projects, however, this type of solution requires a large amount of time and effort on the part of the development team. In this way, it is a better alternative constructive metamodel that allows the abstraction of the general characteristics of a particular context so that the transformation processes.

The model transformation chains are the result that was decided to obtain as a metamodel from a series of transformations, which, being sequential and using the output of the previous transformation as input (under the action of transformation models derived from a transformation metamodel), which will allow obtaining a final element that in the case raised is a UML documentation model.

The use of MDE and specifically model transformation chains differs from reverse engineering. Reverse engineering, also known as return engineering, works backward, part of the final result, divides the product, and performs analysis and measurements to recover the original design information. In the approach of this project with MDE, the start is made from zero and forward, for the generation of UML documentation as a starting point and design of the construction of a product.

Figure 1 is the simplest representation of a model transformation chain, where:

- MM: Metamodel
- m: Model
- MMt: Metamodel transformation
- mt: Model transformation

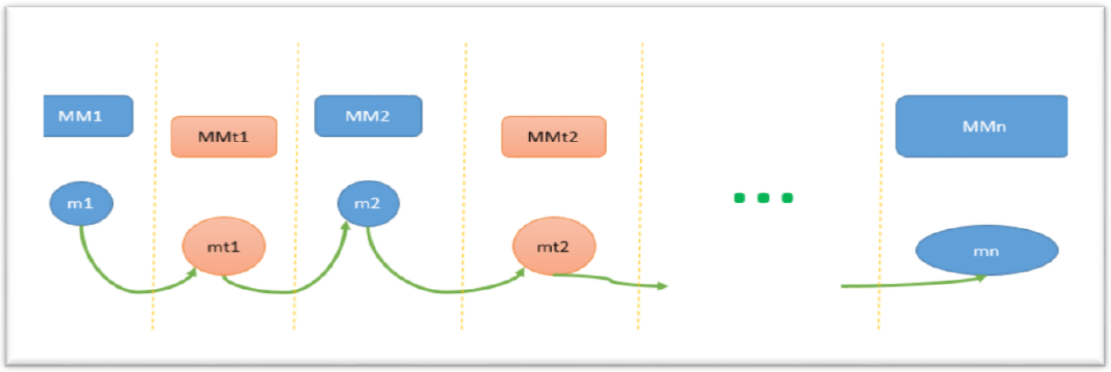

Figure 1: Representation of a model transformation chain

3. Documentation with UML

In the field of software construction, the documentation of projects has become very important, so much so that it is currently considered part of the applications. The generation of the documentation houses several features, from the methodology used for its implementation to the syntax, notation, and visualization with which it is represented.

The importance of documentation lies especially in the fact that what is expressed with the diagrams and mechanisms used must be totally consistent with the totality of the project it represents, referring to any aspect involved in its development. This will involve the entire context of infrastructure, technology, methodology, framework, and architecture related to the represented system.

Additionally, the representation by means of diagrams allows the reuse of potential applications as part of a larger system. For example, if the construction of user authentication and registration system is required, the schematization of this requirement will be equally valid for a banking system, a health care system, or an academic system.

UML has become a standard language that allows modeling all the components of the software construction process. UML does not intend to establish a standard development model, what it really intends is to provide only a modeling language. In UML, the development processes are structured according to the different work domains [12]; for example, the process cannot be the same in the creation of an application executed in real-time as the process to develop an application that is built for management.

The unified modeling language allows showing systems at all levels of abstraction, so that users, leaders, and programmers understand all the features of the application. The models allow to establish better communication with the user for several reasons:

- They show the client an approximation of what the final product will be.
- They provide a first approximation to the problem that allows visualizing how the result will look.
- Show systems at all levels of abstraction.

More and more users use UML for its semantic similarities with other methods (Booch, OMT, OOSE, etc); however, the success of UML is its use in successful projects and the increased need for support tools. and documentation for software.

Although the UML provides a concrete language, it is not the limit to future modeling enhancements. Different techniques have emerged that have gained importance such as cloud development technologies, architecture-based engineering, and Model Driven Engineering; however, other techniques in the future are expected to impact new versions of the UML. Currently, the UML is potentially the basis for many tools, among which we have tools for visual modeling, tools for simulation, and tools for running development environments. UML and its close relationship with object-oriented programming allow the establishment of different methodologies for software development according to new technologies. Currently, there is talk of component-based development and research into its implementation, as a complement to current programming paradigms. If the use of components tends to increase, it would not imply that these techniques can totally replace the current and already established ones.

3.1. UML Model

A model is a conceptual structure of all the topics related to a specific problem. [11] A model is a set of statements about a system under study, these statements are related to the abstraction of an element that can be considered true or false. In software engineering, a model refers to an artifact developed in a modeling language, which describes a system through different types of diagrams [13]. For the current project, the domain given by UML establishes a graphical language to visualize, specify, build, and document a system. The UML provides a standard for describing a "blueprint" of the system, including conceptual aspects such as processes, and system functions, and concrete aspects such as programming language expressions, database schemas, and recycled composites.

3.2. UML Domain Model

The domain model is created to represent the vocabulary and key concepts of the problem domain. It also identifies the relationships between all entities within the scope of the problem domain, and commonly identifies their attributes. Finally, it provides a structural view of the domain that can be complemented by other dynamic viewpoints.

In UML the static structure of the objects in a system is presented in Figure 2, where the base structure of the domain model for the UML context is illustrated.

3.3. UML Metamodel

A metamodel contains the analysis, construction, and development of applicable and useful schemes, rules, constraints, models, and theories for the modeling of predefined classes of problems. It is a global representation that defines how all models are represented. Basically, it marks a syntax about how the elements of the domain and their relationships should be defined in the representation levels of lower abstraction. The metamodels contain the concrete specification of an application domain that follows the syntactic restrictions (an example would be E/R diagrams).

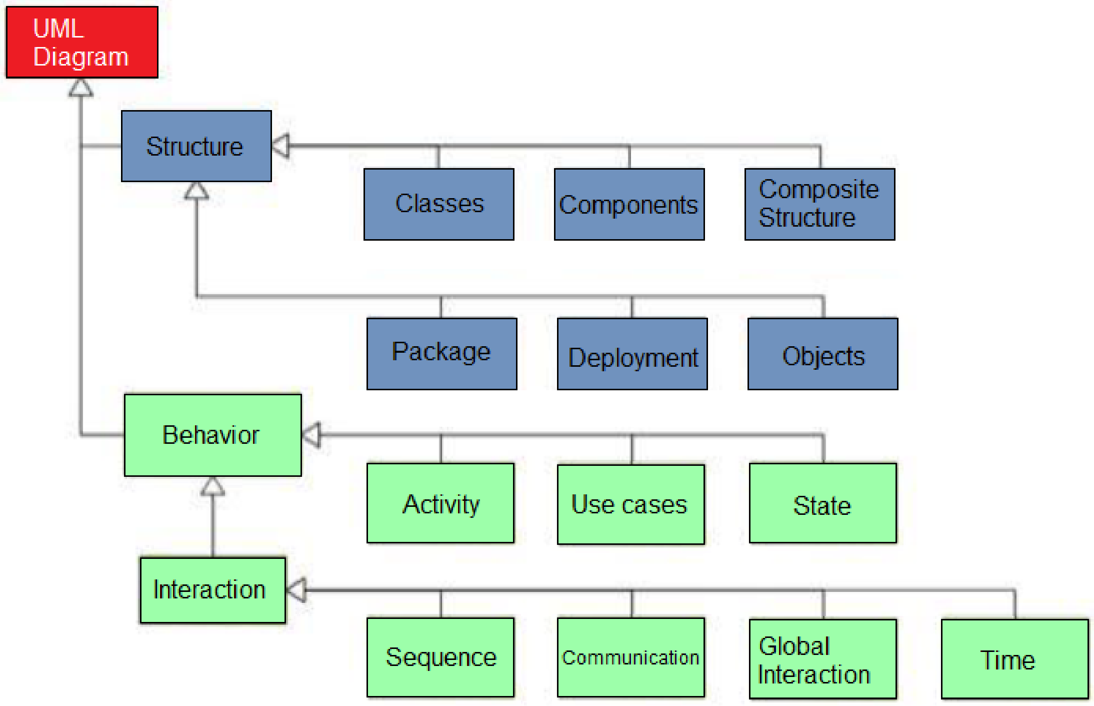

Figure 2: UML Domain Model Structure

The graphic representation of the domain model for the specific domain worked on in the current project is presented in Figure 3 representing the general components required for the generation of the domain metamodel worked on for the generation of UML documentation using transformation chains.

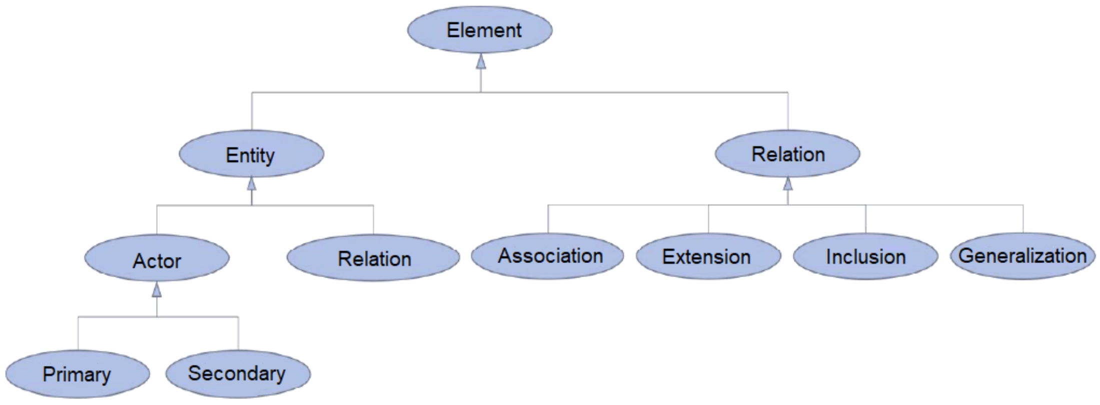

Figure 3: Component Design of the UML Domain Metamodel

4. Proposed Approach

The purpose of Model Driven Engineering (MDE) is to allow the representation of reality through models, which facilitates the implementation of information systems. However, as already mentioned, despite being a fundamental part, the generation of the documentation of the code and components involved in its construction is not given importance, that is where the model transformation chains are an alternative to take. control, due to the possibility they offer to propose a pattern that identifies an input model and converts it into an expected output model. So, it is possible to obtain different solutions not only for an application but also the representation of the whole system, which in our case are UML diagrams.

Using the Model Driven Engineering alternative and through sequential transformations, it is possible to represent different problems raised and abstracted from reality. We can obtain a final product through a model transformation chain to represent the problems mentioned, which are basically the diagrams that make up the UML documentation.

The generated representation is the basis for the construction of systems that will solve the problem initially established and modeled, regardless of the technology being worked on, regardless of whether it is defined and established or whether it is a new and experimental one.

The mechanism that we decided to implement to carry out a series of sequential transformations is the model transformation chains, using as input, the output of a previous transformation (by means of transformation models derived from a transformation metamodel), which will allow obtaining a final element that in the case raised in this project is a UML documentation model, as presented in Figure 4. The most important thing is that the result is an exact and understandable representation both for the user who requests the requirement and for the user developer engineer who performs software coding with the solution to the requirement.

Figure 4 is the representation with a conceptual map of the software development cycle, where different factors intervene during each of the stages. In this conceptual map, each of the elements that can intervene is reflected at a general level, emphasis is placed on the highlighted part in red, they are the components related to the execution of the project for the generation of UML documentation through transformation chains. of models.

4.1. Metamodel implementation

For the continuation of the project Generation of UML documentation through model transformation chains, the use of EMF for the construction of the domain metamodel is proposed. EMF (Eclipse Modeling Framework)[2] is a modeling framework with easy code generation for building tools and other applications based on a structured data model. From a specification of the model described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, a set of classes that allow command-based viewing and editing of the model, and a basic editor. Models can be specified using Java Annotation, XML documents, or modeling tools such as Rational Rose.

Generation of UML documentation through model transformation chains to be imported into EMF. Most important of all, EMF provides the foundation for interoperability with other EMF-based tools and applications.

[2]https://www.eclipse.org/modeling/emf/

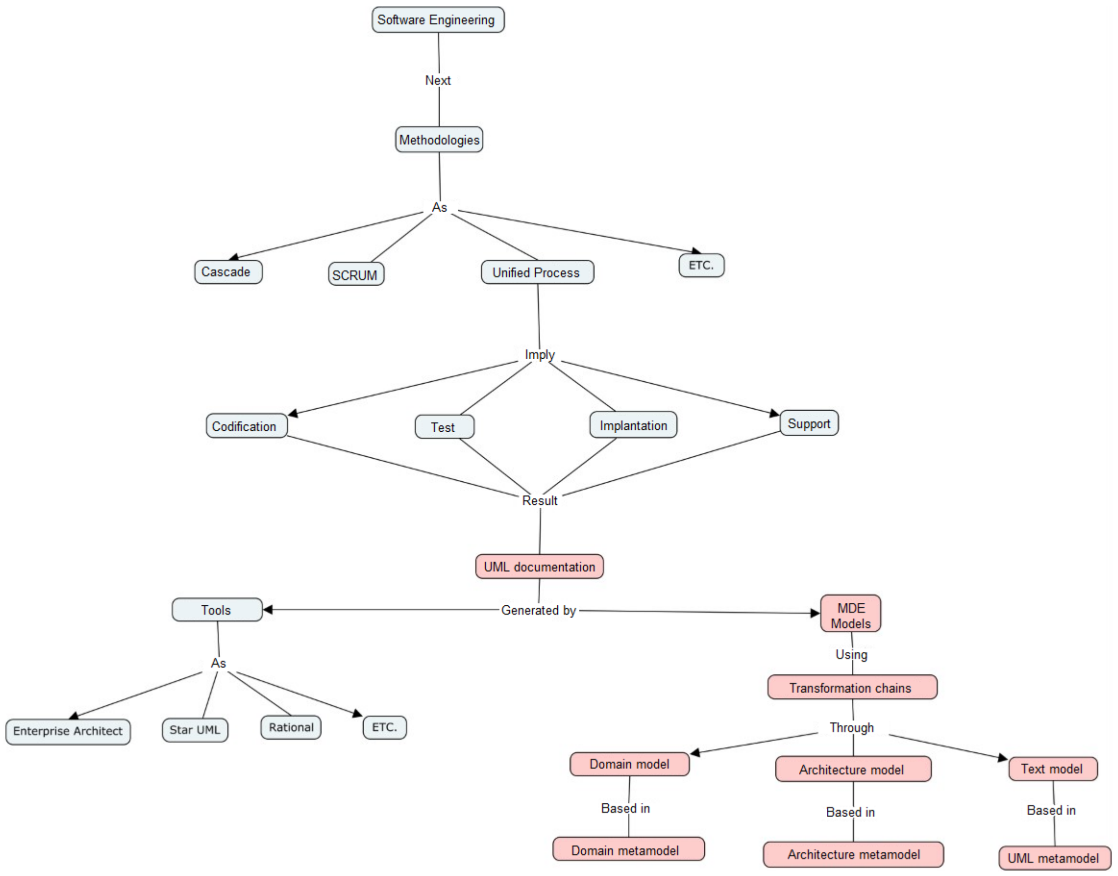

Figure 4: Conceptual map of the use of model transformation chains to generate UML documentation

The Eclipse Modeling Framework, better known as EMF, is the framework provided by Eclipse for the creation of models and metamodels that facilities for automatic code generation and for the construction of model-based tools [14].

EMF allows automatically generating Java classes that implement elements of our models, as well as adapter classes that allow modelers to view and edit a model.

The main functionalities of EMF are:

- Design Ecore metamodels, with two types of editors:
  - Editor based on a tree-like structure
  - Visual editor, like a UML class diagram
- Build model editors based on a tree-like structure, EMF allows generating Java classes that support the metamodel and classes for testing with JUnit.

The EMF core (EMF core) consists of the following elements:

- EMF allows defining metamodels based on Ecore, in this framework, the models can be created and modified from an Ecore metamodel, there is also supported to manage the persistence of models through XMI serialization and in addition to an API for reflection of objects.

- EMF.Edit is a framework that allows generating Java classes to edit an EMF model and consists of:
  - Classes that support editing properties, content providers (content and label provider), and editing Property Sheets (property sheets).
  - A command framework for building an EMF model that includes `undo` and `redo` operations.
- EMF.Codegen is a tool that allows the generation of a complete editor from an Ecore metamodel.

4.2. Ecore Components

Ecore is made up of the following elements (see Figure 5):

- **EPackage:** component that allows organizing classes and data types.
- **EClass:** concepts in the metamodel
- **EReference:** association among concepts
- **EAttribute:** concept properties
- **EDataType:** type of an attribute

Ecore components have the relationships, attributes, and operations presented in Figure 5, which is a diagram created with EcoreTools, which is part of the Eclipse modeling package. Represents Ecore components, references, and attributes.

The image may be complex at first glance; however, it contains detailed and important information. With this diagram, it is possible to obtain and understand the operation of all the components of Ecore only by performing the interpretation of this diagram.

4.3. Ecore Metamodel in EMF

The implementation of the metamodel in EMF with Ecore allows having the starting point of the general objective of the project, which is the realization of a model transformation chain to obtain UML diagrams of an application.

The construction of the metamodel is done in the Eclipse Modeling Tool development IDE. In the tool palette, we add to the diagram the classes required for the object domain model of this project, documentation with UML.

The Ecore file allows you to define the elements detailed below and is presented in Figure 6:

- **EClass:** Element(Object Name), Entity, Actor, Type, Relation
- **EReference:** Reference
- **EAttribute:** Name and description.
- **EDataType:** Figure Enumeration

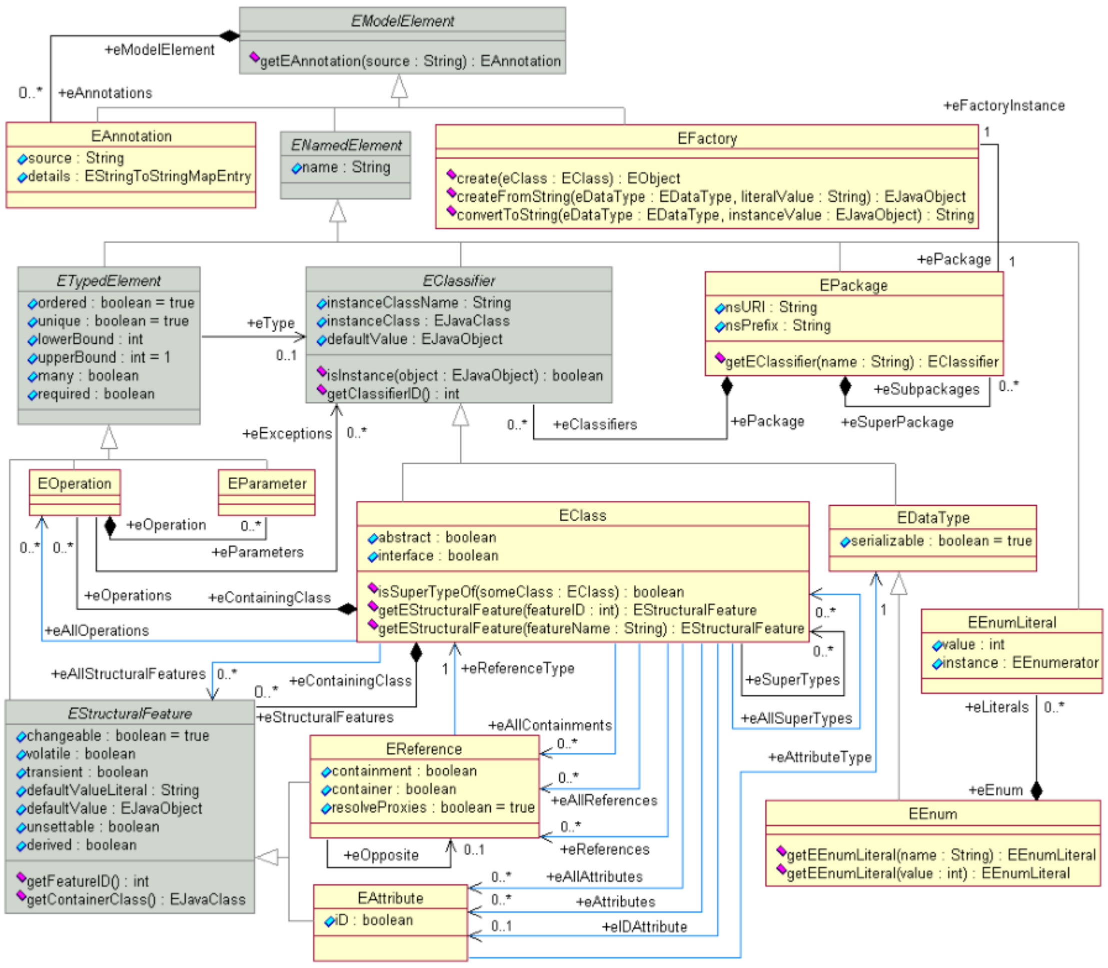

Figure 5: Ecore relationships, attributes, and operations.

5. Conclusions

UML has become a standard language that allows modeling all the components of the software construction process. UML does not intend to establish a standard development model, what it really intends is to provide only a modeling language. In UML, development processes are structured according to different work domains. The unified modeling language allows systems to be displayed at all levels of abstraction, so that users, leaders, and developers understand all the features of the application.

Using the Model Driven Engineering alternative and through sequential transformations, it will be possible to represent different problems raised and abstracted from reality.

The generated result, which is basically the diagrams that make up the UML documentation, would be important because it is an exact and understandable representation both for the user who requests the requirement and for the developer who codified the solution to the problem.

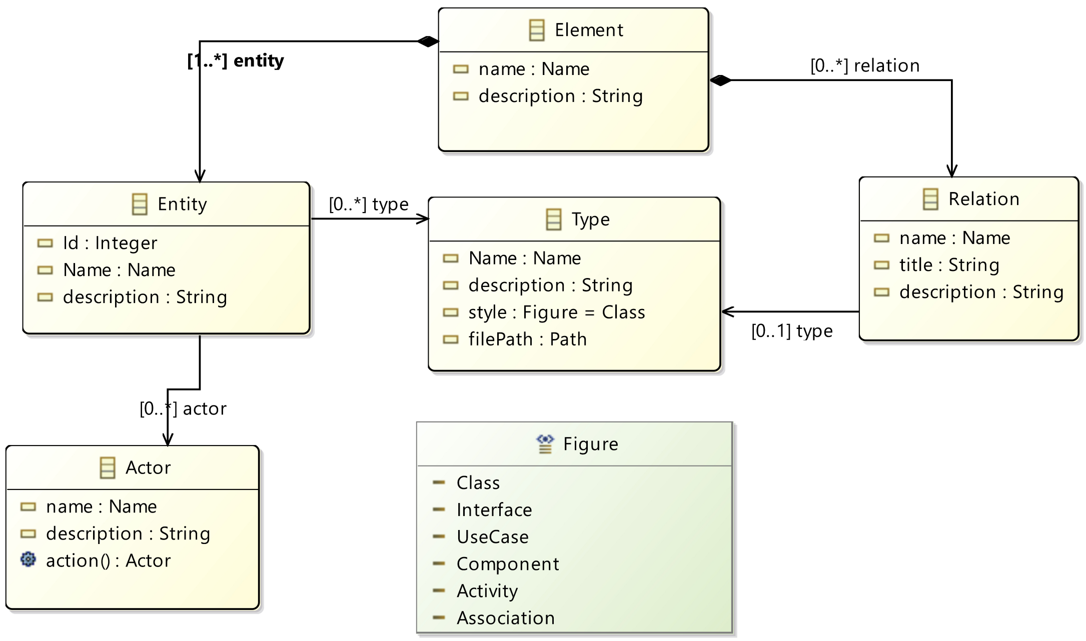

Figure 6: UML Documentation Model in EMF

The level of detail used in the UML diagrams facilitates the construction of the domain model and, consequently, the transformations required for the result of the project in progress.

Likewise, it is important to highlight that much of the semantic information required for the generation of UML diagrams rests on the structure defined in the domain metamodel and the generation templates, so it can be concluded that the platform is not only made up of its model but also by the templates that are specific to each technology.

Modeling the domain metamodel separately uproots elements of the logical and physical views of the specific modeling tool, as the design patterns are generalizable and completely independent of application modeling. Therefore, the same logical view can be used for different application models, or reciprocally, different logical architectures can be used for the same application model, which translates into saving efforts when modeling with EMF.

The implementation of the metamodel in EMF with Ecore allows having the starting point of the general objective of the project, which is the realization of a model transformation chain to obtain UML diagrams of an application. EMF offers a variety of elements that are sufficient to build the core of the context, establishing rules through relationships and attributes. This practical advance will allow following with the investigation to continue with the elaboration of the transformations that are required to fulfill the goal of the final work.

The generation of UML documentation using MDE and applying model transformation chains allows us to have the first input as a starting point in the construction of an application. Additionally, it will work in parallel in the development stage to provide the diagrams that define the correct coding flow of the software required. However, there is the possibility of taking advantage of the power of MDE to implement a project that allows generating the code from the construction of models and metamodels from which a partial generation of source code can be obtained.

References

[1] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Ganapathy, K. Holley, Soma: A method for developing service-oriented solutions, IBM systems Journal 47 (2008) 377–396.

[2] A. Velasco, J. Aponte, Automated fine grained traceability links recovery between high level requirements and source code implementations, ParadigmPlus 1 (2020) 18–41.

[3] A. L. Harris, M. Lang, B. Oates, K. Siau, Systems analysis & design: An essential part of is education., Journal of Information Systems Education 17 (2006).

[4] H. Florez, M. E. Sánchez, J. Villalobos, Embracing imperfection in enterprise architecture models., CEUR Workshop Proceedings (2013) 8–17.

[5] J. Wiegand, et al., Eclipse: A platform for integrating development tools, IBM Systems Journal 43 (2004) 371–383.

[6] D. Gaševic, D. Djuric, V. Devedžic, Model driven engineering and ontology development, Springer Science & Business Media, 2009.

[7] P. Gómez, M. E. Sánchez, H. Florez, J. Villalobos, An approach to the co-creation of models and metamodels in enterprise architecture projects., J. Object Technol. 13 (2014) 2–1.

[8] J. Bézivin, On the unification power of models, Software & Systems Modeling 4 (2005) 171–188.

[9] 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, 2012, pp. 27–32.

[10] J. Ludewig, Models in software engineering–an introduction, Software and Systems Modeling 2 (2003) 5–14.

[11] H. Florez, Model transformation chains as strategy for software development projects, in: The 3rd International Multi-Conference on Complexity, Informatics, and Cybernetics: IMCIC 2012, 2012, pp. 1–10.

[12] G. Engels, R. Heckel, S. Sauer, Uml—a universal modeling language?, in: International Conference on Application and Theory of Petri Nets, Springer, 2000, pp. 24–38.

[13] A. R. Da Silva, Model-driven engineering: A survey supported by the unified conceptual model, Computer Languages, Systems & Structures 43 (2015) 139–155.

[14] H. Florez, M. Sánchez, 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.