UNIVERSIDAD POLITÉCNICA DE VALENCIA
ESCUELA TÉCNICA SUPERIOR DE INFORMÁTICA APLICADA

TÍTULO:

Transformación de Modelos Dirigida por Atributos de Calidad

Autor:
Javier González Huerta

Directores:
Emilio Insfran
Silvia Abrahão

Transformación de Modelos Dirigida por Atributos de Calidad

Valencia, Enero de 2009

En la preparación de este proyecto recibí ayuda y aliento de muchas personas. No tendría suficiente espacio para mencionarlos a todos.
Hay una enorme lista de personas que me han brindado su apoyo y ayuda y de los cuales recibí importantes aportaciones. A todos ellos: GRACIAS

Abstract

Model Transformations are one of the most important artifacts in the Model-Driven Software Development (MDSD) lifecycle. A model transformation is a process of converting one model to another model. A model may be transformed to multiple models that are functionally equivalent but very different with regard to their quality properties. Alternative models are generated since alternative transformations may appear during the transformation process. In the literature, several proposals try to integrate quality in model transformation processes. However, none of them provides a rigorous solution to tackle this problem. Thus, there is a needed to define transformation processes that are able of ensuring some desired properties in the target model.

In this project, we define an architecture for quality-driven model transformations. The main goal of the architecture is to define a set of artifacts (mainly composed of model and metamodels) and a process which allows the definition and execution of model transformations in which the selection of alternative transformations are takenis done based on quality attributes. The rationale of this approach is to be able to automatically select the alternative transformation that an experienced software developer would select if the transformation process were manually applied.

The architecture proposed in this final project is applied in a proof of concept where sequence diagrams from a requirements model is will be transformed into UML class diagrams.

Resumen

Las transformaciones de modelos son uno de los recursos más importantes con los que cuenta el Desarrollo de Software Dirigido por modelos. Una transformación de modelos es el proceso por el cual un modelo puede convertirse en otro nuevo modelo. Un modelo puede ser transformado en varios modelos que, siendo funcionalmente equivalentes son diferentes en con respecto a sus atributos de calidad. Los distintos modelos se generan puesto que durante el proceso de transformación aparecen distintas alternativas, y en función de cuál sea la alternativa seleccionada en cada caso se generara un modelo distinto. En la literatura hay varias aproximaciones que intentan integrar la calidad en los procesos de transformación de modelos, pero ninguna de ellas ofrece una solución rigurosa para atajar este problema. Todo ello hace necesario la definición de procesos de transformación que sean capaces de asegurar que los modelos destino de dichas transformaciones tengan las propiedades deseadas.

En este proyecto definimos una arquitectura para guiar las transformaciones de modelos basándonos en los atributos de calidad. El principal objetivo de esta arquitectura será la definición de los artefactos y procesos necesarios para poder diseñar transformaciones de modelos en los que las selección de transformaciones alternativas se lleva a cabo basándose en atributos de calidad. La base de esta propuesta es que automáticamente se puedan seleccionar las mismas transformaciones alternativas que un desarrollador experto seleccionaría si el proceso fuese llevado a cabo de forma manual.

La arquitectura propuesta en este proyecto fin de carrera será aplicada a un caso de estudio donde se transformarán diagramas de secuencia, extraídos de la especificación de requerimientos de un sistema real en diagramas de clase UML, y podrá verse como seleccionando unos u otros atributos de calidad, el proceso de transformación cambia y los modelos generados tendrán efectivamente esos atributos de calidad.

Index

List of Illustrations:

List of Tables

1. Introduction

Model Driven Software Development (MDSD) is becoming one of the most important approach in the software development community. MDSD is about using models and model transformation to produce software. Those model transformations are used in every step in the lifecycle. Nowadays the transformation process is made in a deterministic way, in the sense that only works with source and target models and no additional information is taken into account to choice between alternative transformations. Those transformation processes can be enriched with other kind of information. We have put our focus in quality attributes. Alternative transformations should be present in the transformation process. The selection of an alternative transformation could improve or worsen the quality properties of the output artifact. Our proposal is to decide what transformation will be applied not only taking care of the constructs in the source model but also the quality attributes we want to maximize in the target model.

The main goal of this project is to define an architecture for Quality Driven model transformations which will allow guiding transformation processes based on Quality attributes.

1.1 Problem Statement

In this work, we focus on one of the activities in MDSD: transforming models. A model transformation is a process of converting one model to another model. In a transformation process a source model can be transformed into more than one different target models that are functionally equivalent but still differing in the quality properties they possess. Alternative models are generated since alternative transformations may appear during the transformation process. If the software engineer want to compare functionally equivalent alternative models from various quality perspectives then alternative transformations must be identified. Generation and evaluation of alternative transformations must be explicitly addressed in a Model-Driven Development (MDE) process. Once those alternative transformations have been detected, the choice between them could be done based on multiple quality attributes.

OMG in (OMG, 2002) *proposed that "A whole range of quality of service requirements can be used to guide transformations" and "In a transformation to a PIM, specific transformation choices will be made according to the particular qualities required at each conformance point in the model".* Although there were no additional proposals made by OMG related to how include those aspects in the MDA transformation processes and how this quality information should be represented and used inside model transformations.

There are many proposals in the last few years trying to use quality criteria to drive transformations, but none of them define how the quality information should be associated

with the different alternative transformations and how quality information should be represented to be taken into account in transformation processes. There was a need of a systematic approach that defines how the quality can be included in MDA processes.

We have developed and architecture for selecting alternative transformations based on quality attributes.

1.2 The Approach

To cope with this problem, we propose an architecture which will allow us guiding the transformation process based on quality attributes. With this guidance we will be able to obtain target models in which selected quality attributes are present.

The transformations process could require some extra artifacts to cover the extra information needed. This is why we have defined a multimodel transformation in the sense that there will be multiple input models and one output model. The extra models are by one side a model for expressing the quality attributes to be maximized and by the other side a model to express the relationships between quality attributes and alternative transformations.

The transformation process is divided in two phases: i) *rule analysis*, where the domain expert identify the alternative model transformations and relate them to quality attributes. The goal is to establish how the model transformations of a particular domain will impact the product quality. This can be done by using empirical evidence gathered in controlled experiments. This phase also deals with a trade-off analysis among quality attributes; and ii) *transformation phase*, which executes the model transformations for ensuring the quality of the products obtained.

The proposed architecture offers a flexible model-driven development process where the model transformations are selected depending on the desired qualities of a particular target model. Most important, the process ensures the quality of the generated products. This provides an improvement to the current model-driven development practices where alternative model transformations and their impact on the product quality are not taken into account.

Once the architecture have been defined we worked with an specific domain, identifying alternative and non alternative transformations and defined those transformations in terms of transformation rules.

Finally, to demonstrate the feasibility of the proposed architecture we perform a case study of model transformations guided by quality attributes.

1.3 Project Context

This work have been conducted as part of the following research projects of the Software Engineering and Information Systems (ISSI) research group from the Department of Computer Science (DSIC) at the Universidad Politécnica de Valencia:

- Integration of Quality in Model-Driven Software Development (CALIMO)
  Granted by *Generalitat Valenciana*, *Conselleria de Educación* - GV/2009/103
  Duration: 01/01/2009 until 31/12/2009
  Main researcher: Silvia Abrahão
  Number of researchers: 5

- Quality-Driven Model Transformations Granted by Universidad Politécnica de Valencia, with reference UPV- 20080008.
  Duration: 01/01/2008 until 31/12/2009.
  Main researcher: Silvia Abrahão
  Number of researchers: 5

This project was developed with the support of a scholarship founded by the Universidad Politécnica de Valencia in the context of the Quality-Driven Model Transformations project.

1.4 Outline of the final project

The remainder of this final project is organized in the following chapters:

Chapter 2 describes the concepts used in this final project. It introduces the notion of Model-Driven Architecture and Model-Driven Engineering. Furthermore, it defines the concepts of model, , metamodels, and model transformations. Finally, it describes the technological space in which this project has been developed, making a brief description of Eclipse, QVT, and Medini QVT.

Chapter 3 describes the state of the art in Quality-Driven Model Transformations. It discusses the strengths and weaknesses of the research works in this area and the need for an architecture to cope with the problems related to the existing transformation processes.

Chapter 4 proposes a quality-driven model transformation architecture. It gives a first description of the whole process and describes our main design decisions. Furthermore, it describes the main parts of the architecture, the strategy of the processas well as the transformation rules granularity.

Chapter 5 illustrates the application of the proposed architecture to a specific domain: a set of transformation rules to obtain UML Class Diagrams from a Requirements Model. We first identify the different transformations (including the alternative transformations) and then we define the transformations in terms of transformation rules.

Chapter 6 demonstrates the applicability of the architecture described in Chapter 4 and using the specific domain described in chapter 5 in the context of a real system.

Chapter 7 presents our conclusions ,an evaluation of the contributions in this final project, and the directions for future works.

2. Foundations

2.1 Introduction

In this section, we describe the concepts, standards and tools related to the work presented in this final project.

2.2 MDE

Model Driven Engineering MDE Is a discipline of Software Engineering that relies on models as first class artifacts and that aims to develop, maintain and evolve software by means of model transformations.

The Modeling idea is not new, is present in software development processes since many years, and it is used to document software's inner structure. Those models were seen as a merely documents that must be fulfilled during the project lifecycle or as reverse engineering tools for source code visualization. Sometimes models are informal, meaning that they cannot be machine-processed. Programmers use them as guidelines and specifications, but not as something that directly contributes to production. Consequently many view them as peripheral to the production process.

MDE offers a more effective approach; models are parts of the software, have the same meaning of the bulk of the final implementation can be generated from them (Völter, et al., 2006). Models also raise the level of abstraction and improve the quality of the development.

A Model-Driven approach requires languages for model specification, defining transformations definition and metamodel description. MDE is embraced by various organizations and companies including OMG, IBM and Microsoft.

The MDE proposal is to use transformations in order to derive from one model to another, and also to produce the final product by means of a model transformation.

2.3 Model Dive Architecture (MDA)

As explained before MDA is included in MDE definition. The MDA standard from the OMG is just a specific incarnation of the Model Driven Engineering.

MDA is about using modeling languages as programming languages rather than merely as design languages. Programming with modeling languages can improve the quality and the speed of software development.

The objective of MDA is to decouple the way that application systems are defined from the technology they run on (McNeile, 2003).

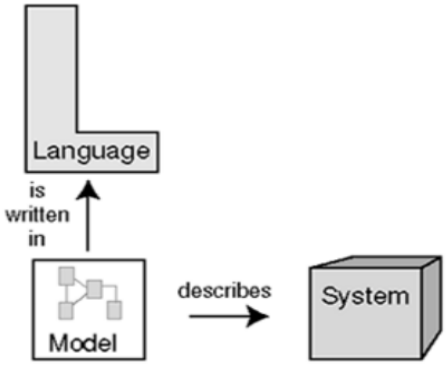


Illustration 1: Model Definition

2.4 Models in MDA

The first class artifacts in Model Driven Engineering are models. The first thing to do is to define what a model is. A model is a simplification (or an abstract description) of a part of the world named system, built with an intended goal in mind (Seidewitz, 2003) (Bézivin, 2001).

A model should be easier to use and understand than the original system, should be able to answer questions of the system. Those answer provided by the model should be the same as those given by the system itself. Models consist of a set of elements with a graphical and/or textual representation.

The idea of MDD is creating different models of a system at different levels of abstraction. Each model represents a given aspect of the system.

According to those definitions, source code is a model too. Source code is a simplified representation of the lower-machine structures and operations that are required to automate the tasks in the real world. Moreover, correct source code is a very useful model since it tells the machine what actions need to be taken to maintain the system's goal.

2.5 Kind of models:

MDA standard (OMG, 2002)defines four kinds of models into the lifecycle of Model Driven Development:

2.5.1 Computation Independent Model (CIM)

A computation independent model is a view of a system from the computation independent viewpoint, which it focuses on the environment and the requirements of the system; structural or processing details of the system are hidden or as yet undetermined. A CIM does not show details of the structure of systems. A CIM is sometimes called a domain model and a vocabulary that is familiar to the practitioners of the domain in question is used in its specification (OMG, 2002). Focuses on the Environment of the system and the requirements the user has on the system; the description provides what the system is expected to do.

2.5.2 Platform Independent Model (PIM)

The platform independent model focuses on the operation of a system while hiding the details necessary for a particular platform. A PIM exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms of similar type. It is also a representation of business functionality and behavior, undistorted by technology details. It shows the part of the complete specification that does not change from one platform to another.

The objective is to postpone in the development process the creation of models that take into account technological aspects of a platform as much as possible. The main advantage is to be able to react efficiently and with low costs to technology changes.

2.5.3 Platform Specific Model

A Platform Specific Model Is a combination of a PIM with additional details of the system's specific platform.

2.5.4 Platform Model

Finally platform models are the representation of technical concepts of platform's parts, the services provided by that platform, and for further use in PSM's, concepts which models the use of the platform by the applications.

2.6 Metamodels

At this level appears a doubt: how to ensure that the structure of a model is correct, how to make the assumptions explicit, automatically checkable etc?.

When modeling is needed to model the structure and well formedness of rules of the language in which de models are expressed. Such models are called metamodel. A precise metamodel is a prerequisite for performing automated model Transformation and for defining accurate models.

A metamodel is a specification model for which the system that describes is models in a certain modeling language. A metamodel says what can be expressed in a valid model of the modeling language.

The interpretation of a metamodel is a mapping of elements of the metamodel to elements of the modeling language. The truth-value of statements in the metamodel can be determined for any model expressed in the modeling language. Since the metamodel is the models specification, a model in the modeling language is valid only if none of these statements are false.

Since a metamodel is also a model, it will be expressed in some modeling language. A metamodel for a modeling language could use the same modeling language. The statements in the metamodel are expressed in the same language as is being described by the metamodel. This is called reflexive metamodel.

2.7 Meta-Object Facility (MOF)

MOF is a standard proposed and defined by OMG (Object Management Group) for supporting MDA. This standard proposes four levels meta-modeling architecture with four metal-layers. Those metalayers are the metamodel of the constructs in the layers above. Each layer is described as follows:

> M3 meta-metamodel layer: In this layer resides the meta-metamodel, a language for defining level M2 metamodels. In OMG standard, MOF is the language defined in this layer.
>
> M2 Metamodel layer: M2 Metamodels are used for describing M1 Models. OMG's UML Metamodel will describe UML constructs.
>
> M1 Model layer: At these level models is where we define models. A model is an instance of a M2 metamodel. I.e., if in M2 layer resides UML metamodel in M1 we could have one of its models: Class Diagram, Activity Diagram, Sequence Diagram and so on.
>
> M0 Instance layer: in this layer objects of the real world are defined.

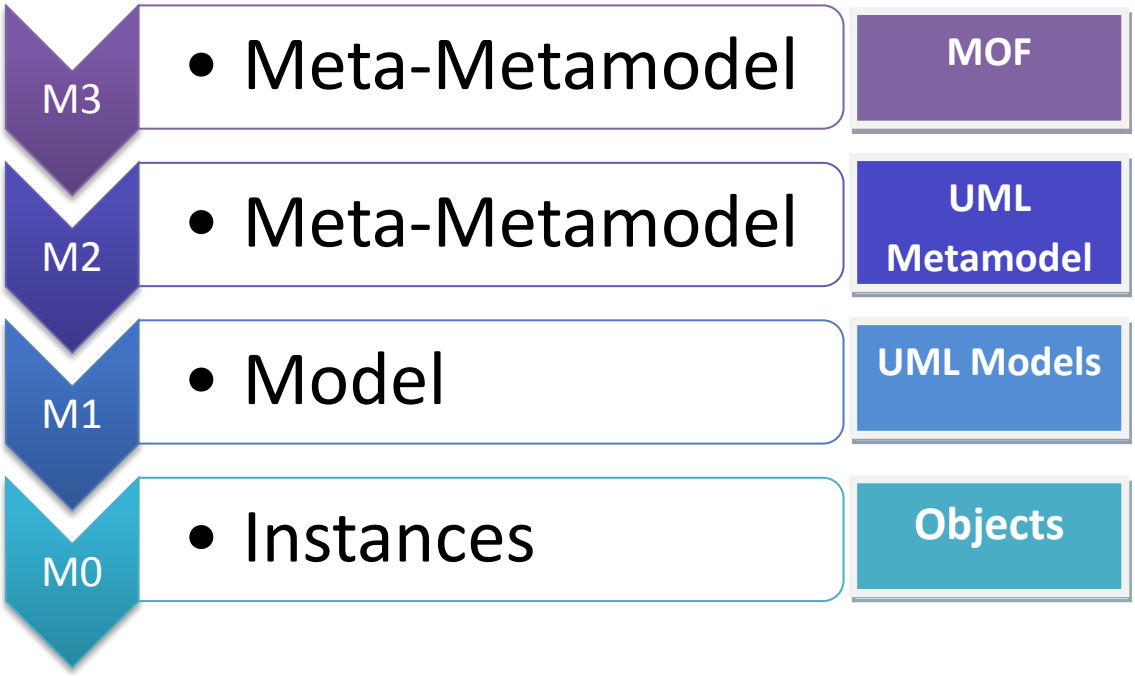


Illustration 2: MOF layers architecture

MOF is closed metamodeling architecture. This means that M3 level could be defined with instances of M3 elements. This implies that whit MOF we can define MOF.

In addition MOF provides concepts to define a language:

- Classes, which model MOF metaobjects.
- DataTypes (property), which model needed descriptive data (i.e., primitive types).
- Associations (property), which model binary relationships between metaobjects.
- Packages, which modularize the models.

OMG has defined two MOF variants:

EMOF for Essential MOF. Is a subset of MOF constructs with the main purpose of providing a framework for mapping MOF models into implementation as JMI or XMI for simpler metamodels.

CMOF for Complete MOF. The CMOF model is used for defining metamodels as UML2. Is built from EMOF metamodels and the UML Core:Constructs.

There are some differences between the UML metamodel to the MOF meta-metamodel which in which the nature of MOF can be observed:

- MOF only supports binary associations while UML supports 'N-ary' associations.
- MOF does not support UML AssociationClasses or qualified Associations. These constructs can be simulated in a MOF metamodel.
- MOF supports the concept of a Reference which allows direct navigation from one Classifier to another. The UML metamodel uses the Target AssociationEnd's for the same purpose.
- Some concepts such as Generalization, Dependency, and Association are reified as classes in UML, but implemented just as Associations in the MOF. (MOF does not require the richness of UML in these areas).

2.8 Model Transformations

Model Transformation: is the process of converting one model to another model of the same system (OMG, 2002). One model may be transformed to several alternative models that can maintain the semantics but with different syntax. The mappings and relations are defined as specializations of transformations.

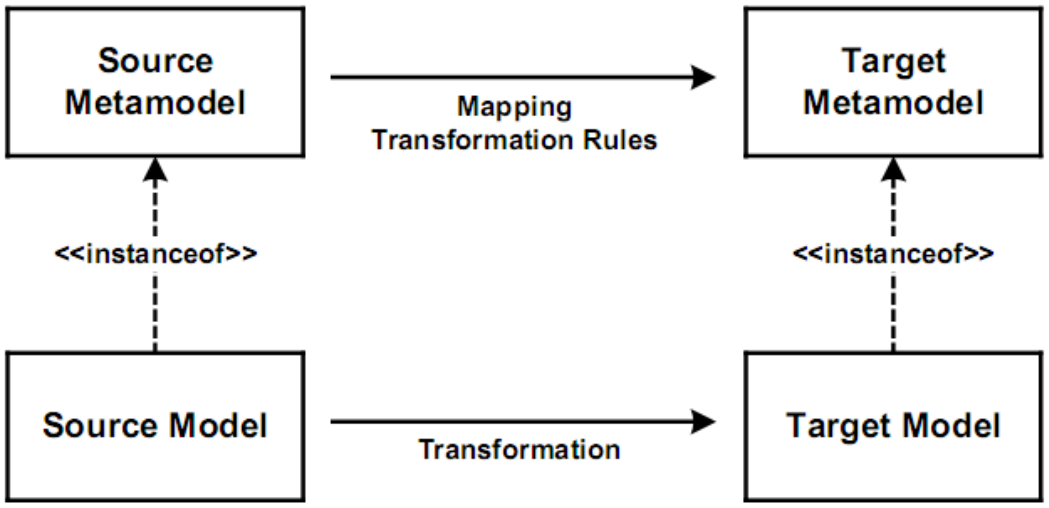


Illustration 3: Model transformation definition

A transformation definition consists of a collection of transformation rules which are unambiguous specifications of the way that (a part of) one model can be used to create a part of another model. The transformations are defined in terms of the metamodels involved in the transformation process. Transformation, transformation definition and transformation rule can now be defined.

> *"A transformation is the automatic generation of a target model from a source model, according to a transformation definition"*
>
> *"A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language"*
>
> *".A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language" (Kleppe, 2003)*

The most important characteristic of a transformation it is the fact that a model transformation should maintain the meaning between the source and the target model. At this point it must be said that the meaning of the model can only be preserved as it can be

expressed in both source and target model. Part of information should be lost if target language is less expressive than source language.

A mapping is defined as a unidirectional transformation in contrast to a relation that defines a bi-directional transformation.

2.9 Model Transformation Taxonomy:

When talking about abstract concepts another way to defining a clarifying terms is to introduce taxonomy as organizing things into groups of thing based on different criteria.

2.9.1 Program Transformations and Model Transformations

Program transformation is a transformation where source and target artifacts are programs (Tom Mens).

Model Transformation is a transformation where source and target artifacts are models. (Tom Mens).

2.9.2 Number of source and target models:

A model transformation should be applicable to a multiple source models and/or multiple target models. (Tom Mens)

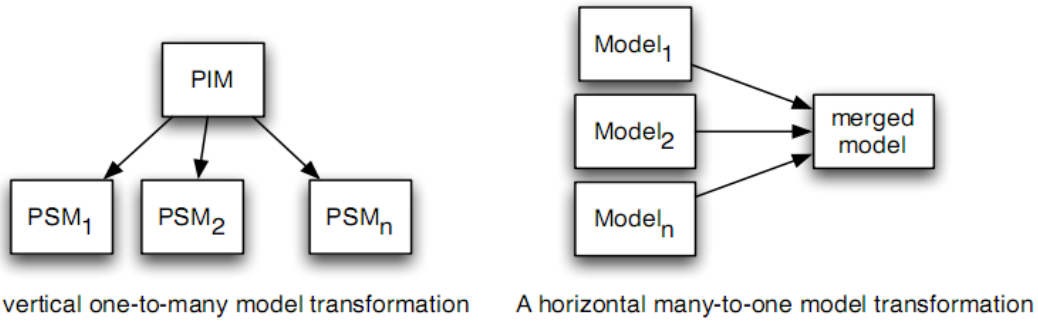


Illustration 4: Different typologies of Multimodel Transformations

Technical Space: A technical space is a model management framework containing tools, mechanisms, techniques and languages and formalisms associated to a particular technology. A technical space is determined by the metamodel that is used (M3 Level). For example W3C promotes the XML technical, which uses XML Schema as meta-metamodel (every model will be expressed in terms of this meta-metamodel) and OMG promotes MDA technical space with uses the MOF as meta-metamodel.

A transformation can work with models belonging to the same or different technical spaces, but in the latter case we need tools and/or techniques to define transformations in the board of both technical spaces. Those techniques can consist in translations from models in one technical space to the other or to have the entire metamodel of one technical space expressed in terms of the metamodel of the second technical space (Tom Mens).

2.9.3 Endogenous and exogenous transformations:

Endogenous transformations are transformations between models expressed in the same language (Tom Mens).

Exogenous Transformations are transformations between models expressed using different languages (Tom Mens).

2.9.4 Horizontal and vertical transformations:

Horizontal transformation is a transformation where source and target models reside at the same abstraction level (Tom Mens).

Vertical Transformation is a transformation between models that reside at different abstraction levels (Tom Mens).

2.9.5 Syntactical versus Semantical transformations:

A syntactical transformation is a transformation that merely transforms the syntax (Tom Mens).

A Semantical transformation is one that takes the semantics of the model into account (Tom Mens).

2.9.6 Mechanisms used for model transformations:

The major distinction between transformation mechanisms is whether they rely on a declarative or an operational (or imperative) approach. Declarative approaches focus on what needs to be transformed into what by defining a relation between the source and target models. Operational approaches focus on the how the transformation itself needs to be performed by specifying the steps that are required to derive the target models from the source models.

On One hand declarative approaches are interesting because some aspects are made implicit in the transformation language as navigation lf a source model, creation of target model and order of rule execution. As such, declarative transformations are easier to write and understand by transformation's domain experts. Declarative approaches also offer services as:

- Traceability in the transformation process: during and after the transformation there will be links between models and its parts.

- Bidirectionality of transformations: A Language or tools that have the property of bidirectionality require fewer transformation rules, since each transformation can be used in two different directions. In Declarative approaches the bidirectionality is assured even when there aren't explicit bidirectional declarations.

On the other hand we have operational (or constructive) approaches that will be useful when Declarative approaches fail to guarantee their services or when the order of a set of transformation needs to be controlled explicitly. Operational approaches will deal with concepts as selections, iterations, sequences and so on (Tom Mens).

2.9.7 Mechanisms used for model transformation

Mechanisms should be interpreted here in a broad sense. They include techniques, languages, methods, and so on. To specify and apply a transformation, ideas from any of the major programming paradigms can be used .One can borrow techniques from the procedural, object-oriented, functional or logic paradigms, or even use a hybrid approach combining any of the former ones.

Nevertheless, it is important to note that a model transformation tool should be dedicated to the construction and modification of models. In this sense, a dedicated domain-specific programming language, preferably even restricted to the technical space of interest, is likely to be more effective than a full-fledged general-purpose programming language.

The major distinction between transformation mechanisms is whether they rely on a declarative or an operational (or imperative) approach. Declarative approaches focus on the *What* aspect, i.e., they focus on what needs to be transformed into what by defining relations between source and target models. Operational approaches focus on the *how* aspect, i.e., they focus on how the transformation itself needs to be performed by specifying the steps that are required to derive the target models from the source models.

Declarative approaches are attractive because particular services such as source model traversal traceability management and automatic bidirectionality can be made implicit in a transformation language: navigation of a source model, creation of target model an order of rule execution. As such, declarative transformations tend to be easier to write and understand by software engineers.

Operational (or constructive approaches) may be required to implement transformations for which declarative approaches fail to guarantee their services. Especially when the application order of a set of transformations needs to be controlled explicitly, an imperative approach is more appropriate thanks to its built-in notions of sequence, selection and iterations. Such explicit control may be required to implement transformations that reconcile source and target models after they were both heavily manipulated outside the transformation tool.

2.10 QVT (Query/View/Transformation)

QVT is a Model Transformation standard defined by the Object Management Group and related to the MOF architecture.

The QVT specification (OMG, 2008) (OMG, 2008) (OMG, 2008) (OMG, 2008) has a hybrid declarative/imperative nature, with the declarative part being split into a two-level architecture. The two-level architecture of the declarative part forms the framework for the execution semantics of the imperative part.

2.10.1 QVT Two Level Declarative Architecture

The declarative parts of this specification are structured into a two-layer architecture. The layers are:

- A user-friendly Relations metamodel and language that supports complex object pattern matching and object template creation. Traces between model elements involved in a transformation are created implicitly.

- A Core metamodel and language defined using minimal extensions to EMOF and OCL. All trace classes are explicitly defined as MOF models, and trace instance creation and deletion is defined in the same way as the creation and deletion of any other object.

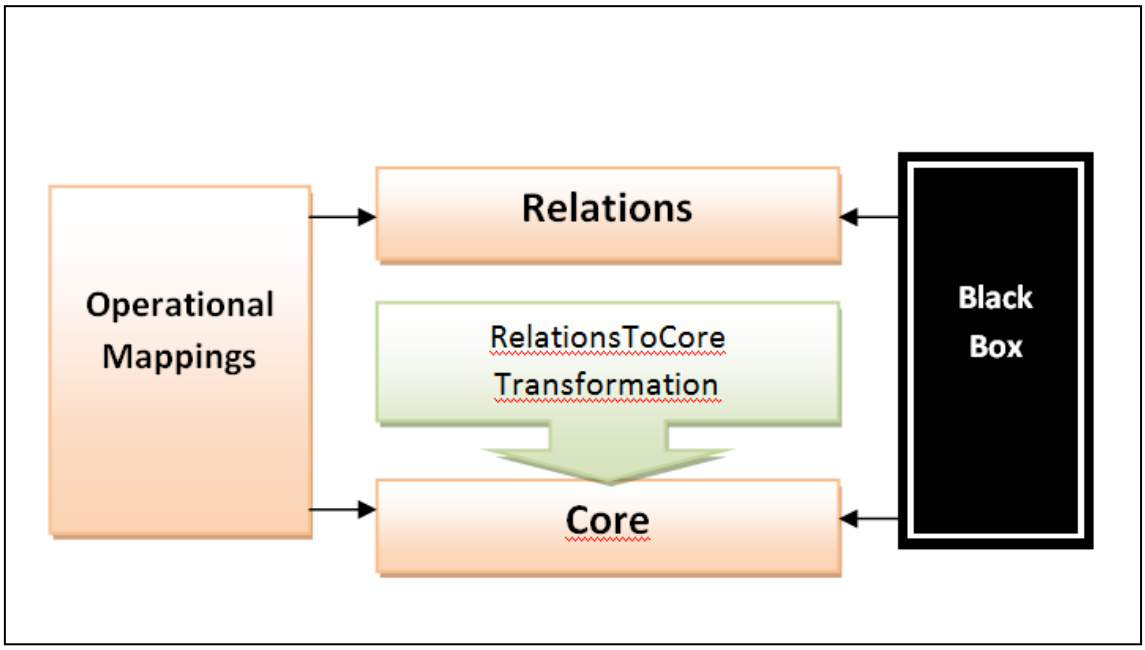


Illustration 5: Relationships between QVT Metamodels

2.10.2 QVT Relations

The Relations language supports complex object pattern matching, and implicitly creates trace classes and their instances to record what occurred during a transformation execution. Relations can assert that other relations also hold between particular model elements matched by their patterns. The semantics of Relations are defined in a combination of English and first order predicate logic, as well as by a standard transformation for any Relations model to trace models and a Core model with equivalent semantics. It can be used purely as a formal semantics for Relations, or as a way of translating a Relations model to a Core model for execution on an engine implementing the Core semantics.

2.10.3 QVT Core

The Core language is a small model/language that only supports pattern matching over a flat set of variables by evaluating conditions over those variables against a set of models. It treats all of the model elements of source, target, and trace models symmetrically. It is equally powerful to the Relations language, and because of its relative simplicity, its semantics can be defined more simply, although transformation descriptions described using the Core are therefore more verbose. In addition, the trace models must be explicitly defined, and are not deduced from the transformation description, as is the case with Relations. The core model may be implemented directly, or simply used as a reference for the semantics of Relations, which are mapped to the Core, using the transformation language itself.

2.10.4 QVT Imperative Implementations

In addition to the declarative Relations and Core Languages that embody the same semantics at two different levels of abstraction, there are two mechanisms for invoking imperative implementations of transformations from Relations or Core: one standard language, Operational Mappings, as well as non-standard Black-box MOF Operation implementations. Each relation defines a class that will be instantiated to trace between model elements being transformed, and it has a one-to-one mapping to an Operation signature that the Operational Mapping or Black-box implements.

This language is specified as a standard way of providing imperative implementations, which populate the same trace models as the Relations Language. It provides OCL extensions with side effects that allow a more procedural style, and a concrete syntax that looks familiar to imperative programmers.

2.10.5 Operational Mappings

Operational Mappings Language can be used to implement one or more Relations from a Relations specification when it is difficult to provide a purely declarative specification of how a Relation is to be populated. Mappings Operations invoking other Mappings Operations always involves a Relation for the purposes of creating a trace between model elements, but this can be implicit, and an entire transformation can be written in this language in the imperative

style. A transformation entirely written using Mapping Operations is called an operational transformation.

2.10.6 The QVT Relations Language

In the relations language, a transformation between candidate models is specified as a set of relations that must hold for the transformation to be successful. A candidate model is any model that conforms to a model type, which is a specification of what kind of model elements any conforming model can have, similar to a variable type specifying what kind of values a conforming variable can have in a program. Candidate models are named, and the types of elements they can contain are restricted to those within a set of referenced packages.

In order to illustrate the definitions we will an example extracted from the QVT specification (OMG, 2008) (OMG, 2008) (OMG, 2008) (OMG, 2008) which shows the relations between elements fron the "UML" domain and "RDBMS" domain.

transformation umlRdbms (uml : SimpleUML, rdbms : SimpleRDBMS) {

In this declaration named "umlRdbms," there are two typed candidate models: "uml" and "rdbms." The model named "uml" declares the SimpleUML package as its metamodel, and the "rdbms" model declares the SimpleRDBMS package as its metamodel. A transformation can be invoked either to check two models for consistency or to modify one model to enforce consistency.

2.10.6.1 Transformation Execution direction

A transformation invoked for enforcement is executed in a particular direction by selecting one of the candidate models as the target. The target model may be empty, or may contain existing model elements to be related by the transformation. The execution of the transformation proceeds by first checking whether the relations hold, and for relations for which the check fails, attempting to make the relations hold by creating, deleting, or modifying only the target model, thus enforcing the relationship.

2.10.6.2 Relation

Relations in a transformation declare constraints that must be satisfied by the elements of the candidate models. A relation, defined by two or more **domains** and a pair of **when** and **where** predicates, specifies a relationship that must hold between the elements of the candidate models.

2.10.6.3 Domain

A domain is a distinguished typed variable that can be matched in a model of a given model type. A domain has a pattern, which can be viewed as a graph of object nodes, their properties and association links originating from an instance of the domain's type.

Alternatively a pattern can be viewed as a set of variables, and a set of constraints that model elements bound to those variables must satisfy to qualify as a valid binding of the pattern. A domain pattern can be considered a template for objects and their properties that must be located, modified, or created in a candidate model to satisfy the relation.

2.10.6.4 When and Where clauses:

A relation also can be constrained by two sets of predicates, a when clause and a where clause, as shown in the example relation ClassToTable below. The when clause specifies the conditions under which the relationship needs to hold, so the relation ClassToTable needs to hold only when the PackageToSchema relation holds between the package containing the class and the schema containing the table. The where clause specifies the condition that must be satisfied by all model elements participating in the relation, and it may constrain any of the variables in the relation and its domains. Hence, whenever the ClassToTable relation holds, the relation AttributeToColumn must also hold Implementations.

```
relation ClassToTable   /* map each persistent class to a table */
   {
          domain uml c:Class {
            namespace = p:Package {},
            kind='Persistent',
            name=cn
        }
       domain rdbms t:Table {
              schema = s:Schema {},
               name=cn,
               column = cl:Column {
                     name=cn+'_tid',
                     type='NUMBER'},
               primaryKey = k:PrimaryKey {
                      name=cn+'_pk',
                      column=cl}
        }
       when {
          PackageToSchema(p, s);
       }
       where {
          AttributeToColumn(c, t);
       }
   }
```

The when and where clauses may contain any arbitrary OCL expressions in addition to the relation invocation expressions. Relation invocations allow complex relations to be composed from simpler relations.

2.10.6.5 Top-Level Relations

A transformation contains two kinds of relations: top-level and non-top-level. The execution of a transformation requires that all its top-level relations hold, whereas non-top-level relations are required to hold only when they are invoked directly or transitively from the where clause of another relation.

```
transformation umlRdbms (uml : SimpleUML, rdbms : SimpleRDBMS) {
       top relation PackageToSchema {…}
```

```
        top relation ClassToTable {…}
        relation AttributeToColumn {…}
}
```

A top-level relation has the keyword top to distinguish it syntactically. In the example above, PackageToSchema and ClassToTable are top level relations, whereas AttributeToColumn is a non-top-level relation.

2.10.6.6 Check and Enforce

Whether or not the relationship may be enforced is determined by the target domain, which may be marked as checkonly or enforced. When a transformation is enforced in the direction of a checkonly domain, it is simply checked to see if there exists a valid match in the relevant model that satisfies the relationship. When a transformation executes in the direction of the model of an enforced domain, if checking fails, the target model is modified so as to satisfy the relationship, i.e., a check-before-enforce semantics.

In the example below, the domain for the "uml" model is marked checkonly and the domain for the rdbms model is marked enforce.

```
relation PackageToSchema   /* map each package to a schema */
   {
      checkonly domain uml p:Package {name=pn}
      enforce domain rdbms s:Schema {name=pn}
   }
```

If we are executing in the direction of uml and there exists a schema in rdbms for which we do not have a corresponding package with same name in uml, it is simply reported as an inconsistency - a package is not created because the "uml" model is not enforced, it is only checked.

However, if we are executing the transformation umlRdbms in the direction of rdbms, then for each package in the uml model the relation first checks if there exists a schema with same name in the rdbms model, and if it does not, a new schema is created in that model with the given name. To consider a variation of the above scenario, if we execute in the direction of rdbms and there is not a corresponding package with the same name in uml, then that schema will be deleted from the rdbms model, thus enforcing consistency in the enforce domain.

These rules apply depending on the target domain only. In this execution scenario, schema deletion will be the outcome even if the uml domain is marked as enforced, because the transformation is being executed in the direction of rdbms, and object creation, modification, and deletion can only take place in the target model for the current execution.

2.10.7 QVT Graphical Syntax

Diagrammatic notations have been a key factor in the success of UML, allowing users to specify abstractions of underlying systems in a natural and intuitive way. Therefore this specification contains a diagrammatic syntax to complement the textual syntax of Section 7.13.1. There are two ways in which the diagrammatic syntax is used, as a way of:

• representing transformations in standard UML class diagrams,

• representing transformations, domains, and patterns in a new diagram form: transformation diagrams.

The syntax is consistent between its two uses, the first usage representing a subset of the second. A visual notation is suggested to specify transformations. A relationship relates two or more patterns. Each pattern is a collection of objects, links, and values. The structure of a pattern, as specified by objects and links between them, can be expressed using UML object diagrams. Using object diagrams with some extensions to specify patterns within a relation specification is suggested. The notation is introduced through some examples followed by detailed syntax and semantics. Illustration 6 specifies a relation, UML2Rel from UML classes and attributes to relational tables and columns. A new symbol is introduced to represent a transformation. The specifications "uml1:UML" and "r1:RDBMS" on each limb of the transformation specifies that this is a relationship between two typed candidate models "uml1" and "r1" with packages "UML" and "RDBMS" as their respective meta models. The "C" under each limb of the relation symbol specifies that both domains involved in this relation are checkonly.

```
relation UML2Rel {
  checkonly domain uml1 c:Class {name = n, attribute = a:Attribute{name = an}}
  checkonly domain r1 t:Table {name = n, column = col:Column{name = an}}
}
```

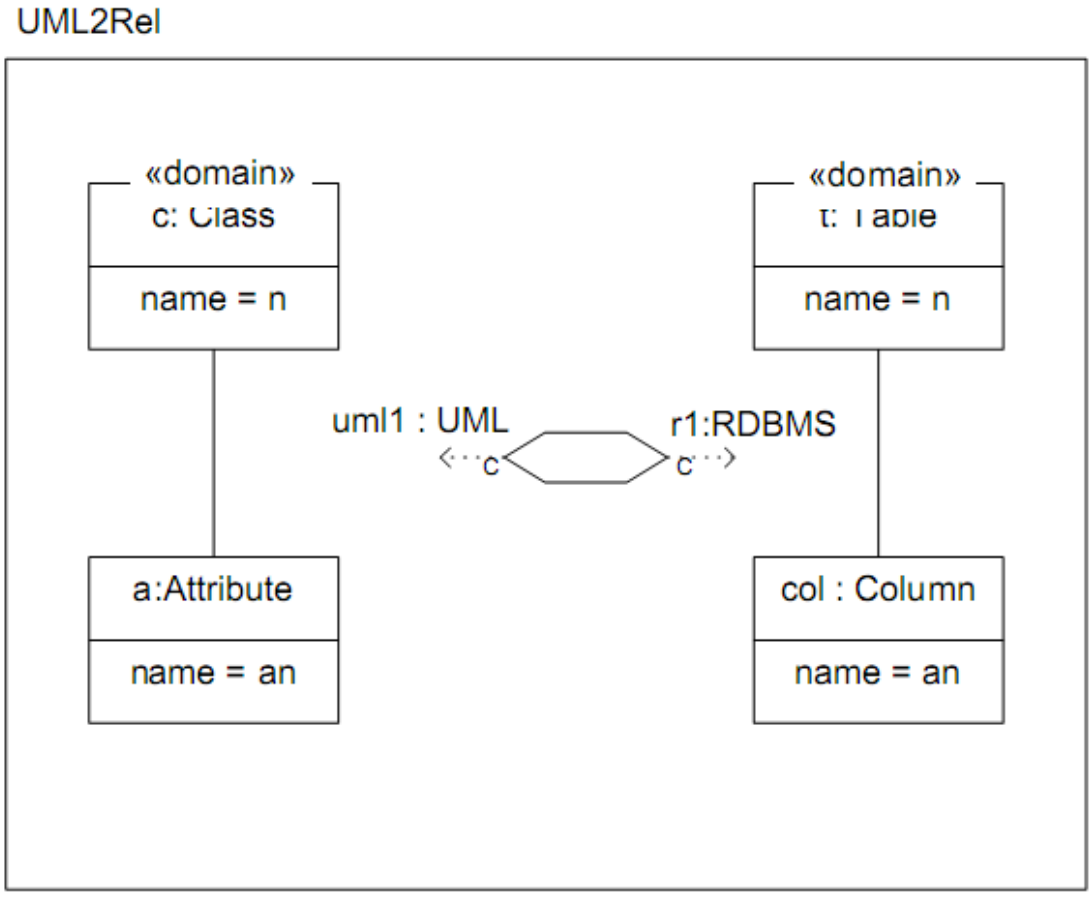


Illustration 6: QVT Graphical notation for UML Class to Relational Table Relation

2.10.7.1 Graphical Notation Elements

Table 1 gives a brief description of the various visual notations elements.

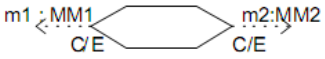
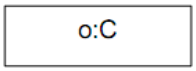
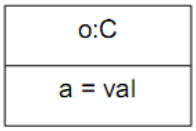
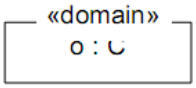
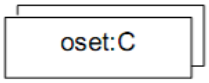
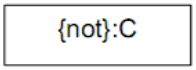
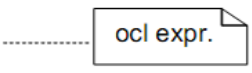
Notation	Description
	A relation between models m1 having MM1 as meta-model and m2 having MM2 as meta-model. The label C/E indicates whether the domain in that direction is checkable or enforceable.
	An object template having type *C* and referred to by the free variable *o*.
	An object template having type *C* and a constraint that the property *a* should take the value *val*. *val* can be an arbitrary ocl expression.
	The domain in a relation.
	*oset* is an object template that matches a set of objects of type *C*.
	A *not* template that matches only when there is no object of type *C* satisfying the constraints associated with it.
	A constraint that may be attached to either a domain or an object template.

Table 1 Diagramic Notations

2.10.8 Implementations

QVT-Operational:

- Borland Together contains implementation of QVT Operational, which have been contributed to Eclipse Foundation and is developed as Eclipse M2M Operational QVT project.
- SmartQVT: an Eclipse open source implementation (Orange Labs) of the QVT-Operational language. QVT compliant and very high potential.
- Eclipse M2M Operational QVT: official Eclipse open source implementation of QVT Operational.

QVT-Core:

- OptimalJ: Early access implementation of the QVT-Core language was in OptimalJ version 3.4 from Compuware. However, OptimalJ has been discontinued.

QVT-Relations:

- MediniQVT: EMF based transformation engine with EPL license for engine and non-commercial license editor/debuger available at http://projects.ikv.de/qvt/wiki.
- Eclipse M2M Declarative QVT: official Eclipse open source implementation of QVT Core and Relations, not yet released.
- ModelMorf: A proprietary tool from Tata Consultancy Services Ltd. Fully compliant with QVT-Relations language. A trial version can be downloaded from http://10.0.2.69:8000/ModelMorf/ModelMorf.htm. The trial version provides a command line utility which consumes and produces models in XMI form. A full-fledged, repository integrated version is available as part of their proprietary modeling framework MasterCraft.

QVT-Like:

- Tefkat : an open source implementation of Tefkat language which is also similar to QVT. High potential. Open source.
- ATL : a component in the M2M Eclipse project. ATL is a QVT-like transformation language and engine with a large user community and an open source library of transformations.
- Model Transformation Framework (MTF): IBM alphaWorks project, last update in 2007.

2.11 Object Constraint Language (OCL)

This section introduces the Object Constraint Language (OCL), as it was defined (OMG, 2003) (OMG, 2003) (OMG, 2003) is a formal language used to describe expressions on UML models. These expressions typically specify invariant conditions that must hold for the system being modeled or queries over objects described in a model. Note that when the OCL expressions are evaluated, they do not have side effects; i.e. their evaluation cannot alter the state of the corresponding executing system.

OCL expressions can be used to specify operations / actions that, when executed, do alter the state of the system. UML modelers can use OCL to specify application-specific constraints in their models. UML modelers can also use OCL to specify queries on the UML model, which are completely programming language independent.

OCL not only can be applied in UML models but also it can be applied into UML or MOF metamodels, since are expressed in UML or a subset of UML. Hereby OCL can be used to restrict metamodel semantics, for example by means of stereotypes or DSL's.

In MDA context, OCL can be used in three ways:

- Precise modeling in MOF M1 level.
- Definition of modeling languages.
- Definition of transformations.

2.12 Technological Spaces

2.12.1 Eclipse

Eclipse is an open source community, whose projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle.

Its most relevant project is The Eclipse Platform. As defined at (Eclipse.org, 2003)The Eclipse platform is designed for building integrated development environments (IDEs) that can be used to create applications as diverse as web sites, embedded Java(TM) programs, C++ programs, and Enterprise JavaBeans (TM).

The Eclipse Platform is built on a mechanism for discovering, integrating, and running modules called plug-ins. A tool provider writes a tool as a separate plug-in that operates on files in the workspace and surfaces its tool-specific UI in the workbench. When the Platform is launched, the user is presented with an integrated development environment (IDE) composed of the set of available plug-ins.

The Eclipse Platform's principal role is to provide tool providers with mechanisms to use, and rules to follow, that lead to seamlessly-integrated tools. These mechanisms are exposed via

well-defined API interfaces, classes, and methods. The Platform also provides useful building blocks and frameworks that facilitate developing new tools.

This modularity is an advantage against other monolithic IDE's where no extra functionalities can be added.

Eclipse is an Open, Flexible and general purpose IDE. For this work we have special interest in Eclipse Modeling Framework (EMF), a subproject of Eclipse. EMF is a framework that allows defining and working with models and is capable of generating code from a model specification.

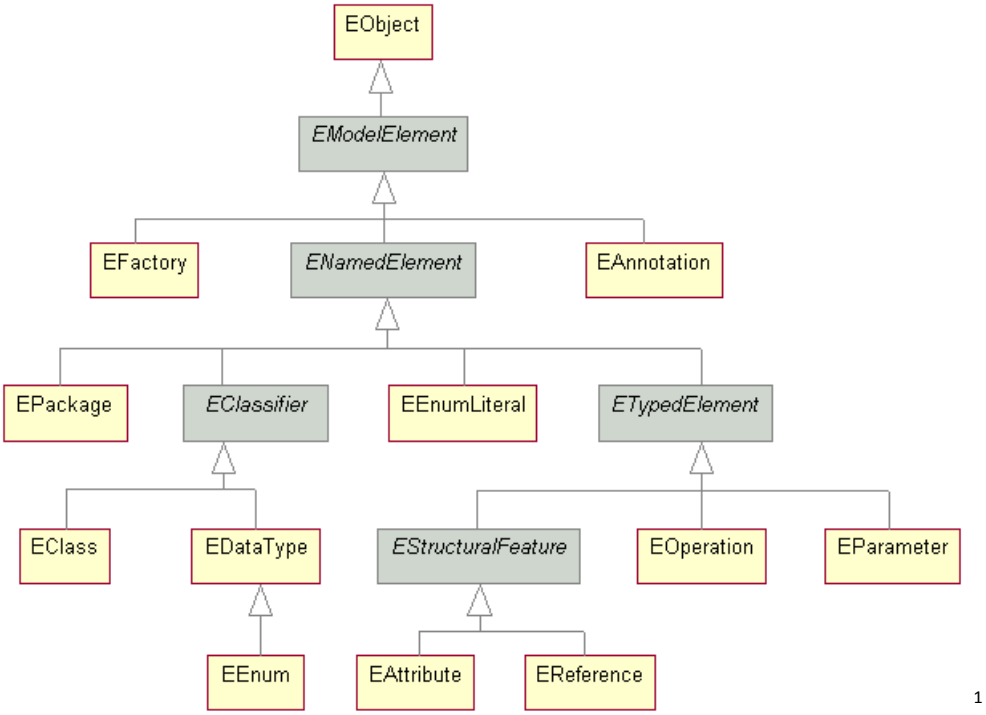
2.12.2 Eclipse Modeling Framework (EMF)

EMF is a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor.

Models can be specified using annotated Java, XML documents, or modeling tools like Rational Rose, then imported into EMF. Most important of all, EMF provides the foundation for interoperability with other EMF-based tools and applications.

EMF started with the Meta Object Facility standard of OMG, and was progressed within the Java Community Process to produce the Java Metadata Interface standard. It is a Java framework and code generation facility for building tools and other applications based on a structured model. EMF provides a mechanism to easily create, save, and restore instances of the classes in your model. This makes it very easy to share data across different applications (Maddeh Mohamed, 2007).

The metamodel based on MOF Core is called Ecore, illustration 7 shows the main components of Ecore.



[1]

Illustration 7: Ecore Components

In the MOF 2.0 proposal the Essential MOD (EMOF), a similar subset of MOF model have been segregated. There are some differences between EMOF and Ecore, mainly at the names of the entities; however EMF Ecore is capable to read and write EMOF.

2.12.3 Medini QVT

Medini QVT is a QVT transformation engine wich implements the OMG's QVT relations specification in a powerful QVT engine. This standard is designed for model-to-model transformations to allow fast development, maintenance and customization of process specific transformation rules.

Highlights among the supported features are:

- Execution of QVT transformations shown the textual concrete syntax of the relations language.
- Editor with code assistant.
- Integrated debugging which allows to run the relations step by step.
- Trace management enabling incremental updates during transformations.
- Key concept enabling incremental updates as well as the transition from manual modeling to automatic transformations.
- Bidirectional transformations.
- Multimodel transformations, multiple source or/and target models.

[1] Figure from The Eclipse Foundation, 2009

There are two ways of working with Medini QVT a standalone version, an Eclipse-like standalone executable or a Eclipse Plug-in. In the standalone version EMF editors and the transformation engine are available for defining transformations, designing models and metamodels. The Eclipse Plug-in integrates QVT relations Engine in the Eclipse IDE. Once the plug-in has been activated the QVT transformations are available as another Eclipse functionality.

The selection of Mediny QVT against other transformation engines for this project is based on:

- Multimodel transformations support.
- Complaint with OMG's QVT Relations standard.
- Bidirectional transformations.
- Eclipse Integration.
- EMF compatible.

2.13 Conclusions

In this chapter we have introduced key concepts to understand the conceptual framework of this project. We have briefly described the paradigms involving this work and the technological spaces we'd used to support it.

Model Driven Software Development is a software engineering approach that promotes the use of models and model transformations as primary artifacts, rising up the abstraction level. This allows improving aspects as interoperability, reusability, maintainability and team productivity. Our work will adopt MDA architecture, the DSDM approach proposed by OMG. This approach comprises the use of models in all the steps of a software development project, until the delivery of the software on a given platform. A MDA development process basically transforms a platform-independent model (PIM) into one or more platform-specific models (PSM), which are transformed into code (code model – CM). The CM is just the actual code generated from PSMs through transformation. Those models are defined well defined with metamodeling techniques and are transformed automatically. The mechanisms introduced by OMG to solve those issues are MOF, UML, QVT, OCL etc.

Focusing on the technological space, this proposal will be based in Eclipse developing. We will use Eclipse Modeling Framework to deploy metamodeling techniques and model definition and the transformation will be defined using QVT relations and the Medini™ QVT Engine.

3. State-of-the-Art

In this chapter we describe the related works dealing with quality of model transformations focusing our attention in approaches trying to guide or drive those transformations based on quality attributes.

In the last few years, some proposals that deal with the quality of model transformations from the perspective of a quality attribute have been proposed.

An organized chronological summary of these studies is presented in Table 1.

Zou and Kontogiannis (2003) proposed a quality-driven reengineering framework for object-oriented migration. Analysis tools, transformation rules, and non-functional requirements for the target migration systems characterize this framework. During the migration process, the source-code transformation rules are associated with quality features of the target system (i.e., coupling and cohesion). This approach was applied to transform a set of GNU AVL libraries into an UML class diagram.

Table 2 Comparison approaches for quality in model-driven development

Proposal	Purpose	Type of Transformation	Input Artifact	Quality attributes	Automation
Zou and Kontogiannis, 2003	Reverse engineering (migration)	Vertical (CM-to-PIM)	Program code	Coupling and cohesion	No
Rottger and Zschaler, 2004	Refinement	Horizontal	Context Models	Response Time	Partial
Merilinna, 2005	Refactoring	Horizontal (PIM-to-PIM)	Architectural models	Performance, availability, reliability, maintainability, modifiability and reusability	Yes
Kurtev, 2005	Synthesis	Vertical (PIM-to-PIM)	UML class models	Adaptability	Yes (Mistral)
Markovic and Baar, 2005	Refactoring	Horizontal (PIM-to-PIM)	UML class models	Syntactical correctness	No
Sottet et al., 2006	–	–	Interface models	Compatibility, error protection, homogeneity-consistency	No
Ivkovic and Kontogiannis, 2006	Refactoring	Horizontal (PIM-to-PIM)	Architectural models expressed in UML	Maintenance, performance and security	No
Kerhervé et al., 2006	Synthesis, refinement	Horizontal and Vertical	Information models	Response time, network delay, network bandwidth	No

(–) means that the proposal does not provide this information

Röttger and Zschaler (2004) proposed an approach for refining non-functional requirements based on the definition of context models and their transformations. This approach has been defined in a software development process that separates the roles of the measurement designer and the application designer. It is the measurement designer's responsibility to specify measurements, context models and transformations among these models. Then, the application designer can apply the transformations when developing a system. Röttger and Zschaler defined a XML-based language for the specification of transformations between abstract and concrete context models.

The transformations used the response time quality attribute Merilinna (2005) proposed a tool for quality-driven model transformations for software architectures. Two types of quality attributes are considered: attributes related to software execution (e.g., performance, availability, reliability) and attributes related to software evolution (e.g., maintenance, modifiability, reusability). The transformations are described according to MDA and a proprietary transformation rule language. The approach only considers horizontal transformations (PIM-to-PIM transformations).

Kurtev (2005) proposed a formal technique for the definition of transformation spaces that support the analysis of alternative transformations for a given source model. This technique provides operations for the selection and reduction of transformation spaces based on certain desirable quality properties of the resulting target model. Specifically, this approach deals with the adaptability of model transformations. To generate the transformation space, the process takes a source model and its metamodel, the target metamodel, and the quality properties as input. The proposal has been applied to a set of transformations to obtain XML schemas from UML class diagrams.

Markovic and Baar (2005) defined a set of transformation rules for the refactoring of UML class diagrams. The rules have been defined using the Query/View/Transformation (QVT) standard of OMG (OMG, 2005). The refactoring is applied to UML class diagrams containing annotated OCL constraints that are preserved when the transformations are applied. Therefore, the syntactical correctness of the target model is preserved.

Similar to this proposal, Ivkovic and Kontogiannis (2006) presented an approach for the refactoring of software architectures using model transformations and semantic annotations. In this approach, the architectural view of a software system is represented as a UML profile with its corresponding stereotypes. Then, the instantiated architectural models are annotated using elements of the refactoring context, including soft goals, metrics, and constraints. Finally, the actions that are most advisable for a refactoring context are applied after being selected from a set of possible refactorings. The proposal has been applied to a case study to demonstrate that the refactoring transformations improve the maintenance, performance and the security of a software system.

Sottet et al. (2006) proposed an approach for model-driven mappings for embedding the description and control of usability. A mapping describes a model transformation that preserves properties. The mapping properties provide the designer with a means for both selecting the most appropriate transformation and previewing the resulting design. A case study that illustrates an application of the mapping metamodel using usability criteria compatibility, error protection, and homogeneity-consistency) was presented. Kerhervé et al. (2006) proposed a general framework for quality-driven delivery of distributed multimedia systems. The framework focuses on Quality of Services (QoS) information modeling and transformations. The transformations between models express the relationships among the concepts of the different quality information models. These relationships are defined in quality dimensions and are used to transform instances of a source model to a target model. Different types of transformations are applied to different layers and services: vertical transformations are applied to transform information between the different layers (user, service, system, and resource), and horizontal transformation are applied to interchange information between services of the same layer.

In summary, some proposals focus on defining horizontal transformations for model refactoring (Merilinna 2005) (Markovic & Baar 2005) (Ivkovic & Kontogiannis 2006). Other proposals are aimed at providing vertical transformations for model refinement (Rottger & Zschaler, 2004), synthesis

(Kerhervé et al., 2006) (Kurtev, 2005), or reverse engineering (Zou &Kontogiannis, 2003). Of these studies, only the one by Kurtev (2005) presents a more systematic approach for selecting alternative transformations according to a given quality attribute.

All these approaches propose quality criteria that can be used to drive the transformations, but very few of these approaches (Kurtev, 2005) (Markovic & Baar, 2005) illustrate them by means of practical examples. With the exception of Markovic and Baar (2005) and Kurtev (2005), the transformations are poorly defined. Therefore, more systematic approaches to ensure quality in MDA processes are needed. Another weakness of these proposals is that they are not empirically validated. The practical applicability of model transformations is reported based on the intuition of the researcher. As pointed out by Czarnecki and Helsen (2006), there is a lack of controlled experiments to fully validate the observations made by the researchers.

Finally, Abrahão *et al.* (2008) had proposed an approach to drive the model transformations based on quality attributes. This approach have been empirically validated in an specific set of transformation rules in order to obtain UML Class diagrams starting from a requirements model. The main objective of the experiments were obtaining empirical evidence about the appropriate rule selection and how this selection can improve the understandability of the obtained UML Class Diagrams. The transformation rules where defined using MOMENT platform. In this work, the authors present a generic architecture for quality-driven model transformations.

Using a controlled experiment empirically validates that alternative transformations can affect to the quality of the output artifacts and uses the information gathered during these controlled experiments as additional inputs of the transformation process. This information will feed the transformation process with the criteria to choose the alternative transformation that maximize the selected quality attribute.

Although the approach could evaluate more than one quality attribute the experiments were performed evaluating the understandability of UML Class diagrams.

Since the architecture proposed was theoretically defined but no infrastructure was developed, the transformations were performed by a transformation engine but no additional models were added to hold the additional quality information and the feedback to the process.

4. Quality-Driven Model Transformations

This Chapter presents an architecture for quality model transformations capable of hold and express transformations between source and target models in which the choice between alternative transformations is made based on quality attributes. In this section we will define the artifacts need to express the extra information required to assure the quality guidance of the transformation process. Moreover the transformation process has been decomposed to allow users and domain experts to interact with the proposed transformation architecture.

4.1 Introduction

In MDSD software development process models are transformed to other models to finally achieve a set of models containing enough details to implement the system. A model may be transformed to multiple models that are functionally equivalent but very different with regard to their quality properties. The selection of a particular model should then be determined based on quality requirements of the output model.

The transformation is executed based on a transformation definition as seen in section 2. Transformation definition contains transformation rules that relate constructs in the source model to the constructs in the target model. Alternative transformations will appear every time a single (or complex) construct in the source model could be transformed into multiple alternative constructs in the target model.

The source and target models are at level M1 according to the MOF terminology and their source and target metamodels are defined in the level M2. The transformation is defined at M2 level and executed at M1 level.

The source model MA is an instance of the source meta-model MMA. Assume that MA contains three elements a1, a2, and a3 shown as rectangles with relations among them. MA has to be transformed to a model that is an instance of the target meta-model MMB. The two meta-models can be used to determine the possible transformations rules. MMA contains two constructs: A1 and A2. MMB contains four constructs: B1, B2, B3, and B4.

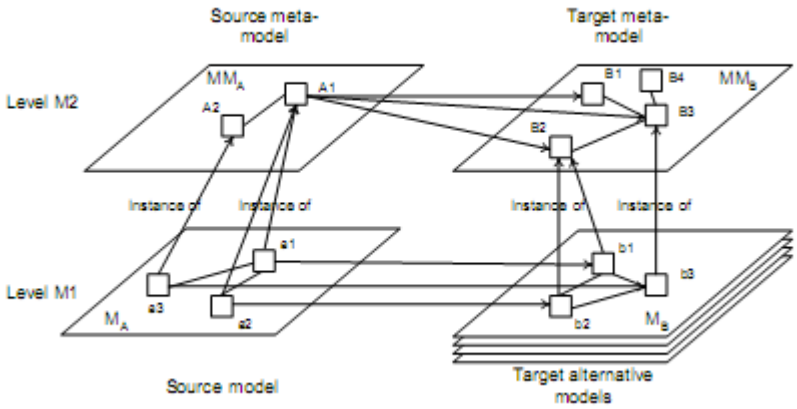


Illustration 8 Alternative transformations for a given source model

Generally for a given construct or set of constructs in the source metamodel there are multiple constructs in the target model to which can be mapped. Assume that the construct A1 in the source meta-model can be alternatively mapped to the three constructs B1, B2, and B3 in the target meta-model. This introduces alternative mappings for each instance of A1 in the source model MA. lustration 8 shows one possible mapping in which a1 is mapped to an instance of B2 and a2 is also mapped to an instance of B2. Other combinations are generally possible and this results in multiple target models. Those alternative target models are functionally equivalent but can be very different with regard to their quality properties. For a concrete problem the software engineer must be able to identify the transformations that lead to a model with the desired quality properties.

This chapter will define a an architecture which will help Domain Experts, starting with alternative transformations, to associate them based with specific quality attributes of the resulting models and to define priorities of those quality attributes on a given Domain. The transformation user will be then able to select the quality attributes to be maximized on the resulting model and executing the transformation to get a model with the selected quality attributes.

4.2 Architecture Overview

In this section we will describe the main components of the architecture and the global process. The domain expert chooses which quality attribute would maximize. He makes the selection based in his experience or previous empirical studies. The quality attributes selection drives the selection of alternative transformations. A quality attribute is a measurable physical or abstract property of an entity (e.g., conceptual model) [(SQuaRE, 2005)

The whole process have been divided in two phases: i) *rule analysis*, and ii) *transformations phase*. Each one has a model transformation process.

Regarding the transformation process, in the first phase a Rule Selection is done. A Quality-Driven Model transformation as could be seen by the user of the transformation, which works with quality, attributes to be maximized and the source and target models.

About the second phase, a model transformation is executed taking a transformation definition as input. A transformation definition contains transformation rules that relate constructs in the source model to constructs in the target model. We use another input for the transformation process that is the definition of the quality attributes together with the corresponding empirical evidence gathered from controlled experiments. This information will feed the transformation process with the criteria to choose the alternative transformation that maximizes the selected quality attribute. The rationale of this approach is to be able to automatically select the alternative transformation that an experienced software developer would select if the transformation process were manually applied.

As seen in the basic definition we will require some extra artifacts to cover the extra information needed. The architecture will be then based on multimodel transformations in the sense that will be more than one input models (at least: input model + the models need to drive the transformation) and just one output.

First one will be a Quality Model. The quality model will have the whole characteristics, sub-characteristics and quality attributes for a given domain. Users will select the attributes to be maximized on the target model.

The next one will be the transformation model for expressing the associations between quality attributes and rules. A MDA expert will be able to change those associations for a given domain, as a result of a feedback after transformation processes.

To model the rules which are applicable in a given transformation (After the user have selected the attributes which wants to improve) it will be needed an active rules model. A user could perform different model transformations using the same active rules set, using different source models, and obtaining target models with the same quality attributes.

4.3 Evolution of the architecture:

Before defining final layout of the architecture several different architectures were considered. Every approach has at least two input models and one output model. The basic input models are the source model and a quality model, where the user selects the quality attributes to be maximized in the target model. The basic output model is the target model of the transformation process. While the architecture was evolving new input models were added.

The first approach was a plain architecture in which each transformation rule is associated with the quality attribute in the rule definition. No additional models where defined; only the source and target model and the quality model. This approach has the advantage of behave correctly in domains in which the association between quality attributes and alternative transformations it is clear enough, and only will be useful for transformation in well-known domains. The associations between rules and quality attributes can't be expressed quantitatively, and this is an ideal and poor realistic approximation. Moreover, this architecture has a lack of maintainability over the rules, every change in the associations among the transformation rules and the quality attributes must be done over the transformation definition (code). In addition the alternatives are not well organized, at one

sight the domain expert can't see the number of alternatives related to a given set of constructs of the source model.

The second approach was an architecture using an additional transformation model. In this approach we have the basic input models and a new model to hold the associations between Rules and Quality Attributes. This approach improves the first approach due to making association Rules -> Quality Attributes more flexible. On the other hand adds complexity to the final rules. The rules have high complexity because have to perform validation against tree models: first the source model to validate that the constructs are present on the source model, and then use the quality and transformation model to see if the user of the transformation has selected a quality attribute which is associated in the transformation model to this rule. Working with the transformation model we save the problem of associating more than one quality attribute to an alternative, and to allow an alternative to be composed by more than a rule in a clear way.

The next approach was architecture with another additional input model, the active rules set. This alternative was so close to the adopted architecture, the main difference was that finally we decide to split the transformation in two steps. The structure of the rules is more or less the same than in the adopted architecture, but then affects to the modularity, rule selection and transformation application.

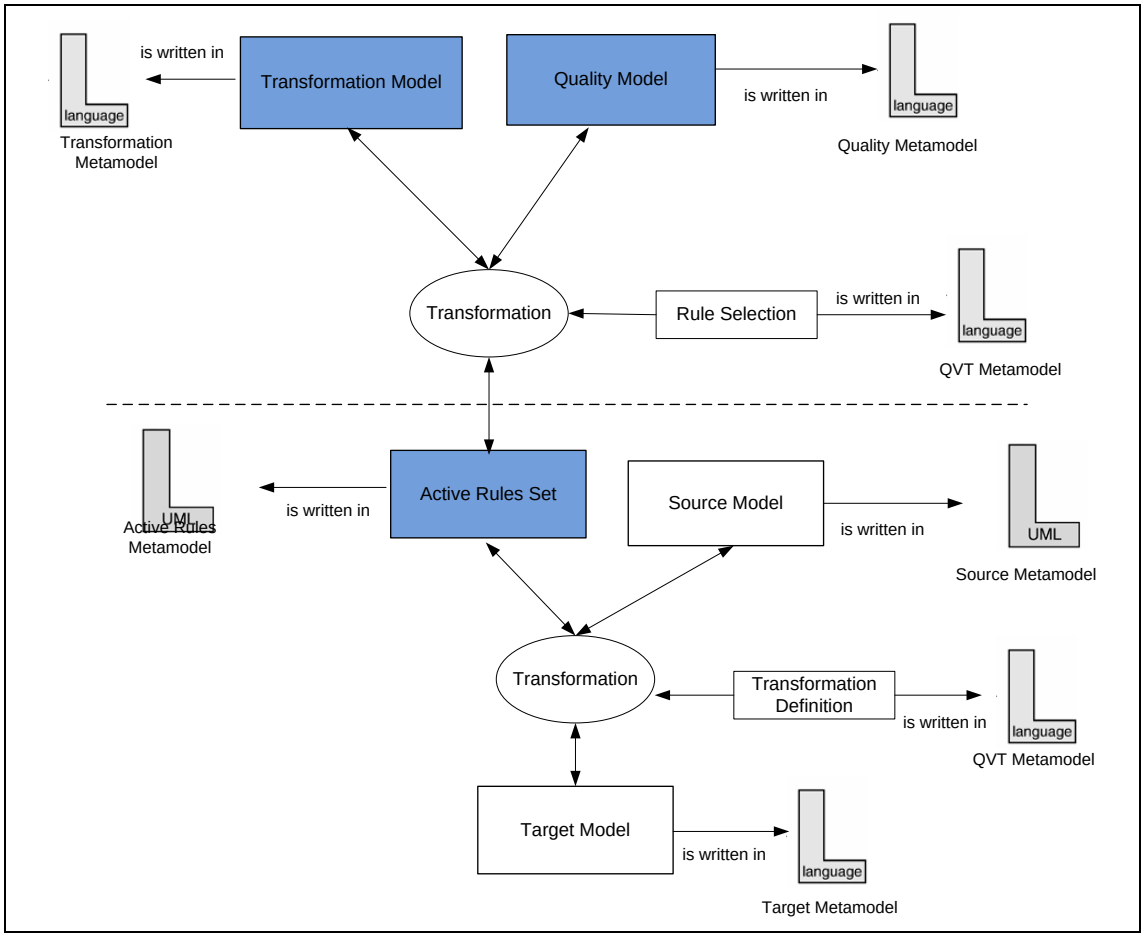


Illustration 9: Transformation schema

Finally we have decided to divide the process into two steps. The first one takes the quality transformation model and the transformation metamodel as inputs and as output generates a Rule Model with the set applicable rules.

With this first step we reach the domain independence. For any kind of source or target language, we can drive the transformation between them based on the quality and transformation model and obtaining a set of rules that will be applied when alternative transformations appear. Only rules related to quality attributes to be maximized will appear in the se of active rules.

The second step will only consider the source model and a set of applicable rules and as output will generate the target model. This will be an instance of the target metamodel and will be built using only the rules selected in the first step in addition to the non alternative rules, which are the rules that are applied whenever the left side of those rules is matched in the source model.

The transformation process is made by means of structural pattern matching on the source model and it's conversion into the target model. When a set of constructs is detected in the source model and alternative transformations are possible for this construct, then the transformation to be applied must be in the active rules set.

4.4 Model Definition:

To achieve the quality guidance of Model Transformations we will need additional input artifacts. as were defined in the section 4.2 were the transformation model, the quality model and the active rules set.

Quality Model: It will be an instance of the generic metamodel proposed at (SQuaRE, 2005), where the software quality can be decomposed into quality characteristics, quality sub-characteristics and quality attributes. With this input model the user can select the quality attributes to be maximized in the output model.

Quality Metamodel: the figure shows the relevant parts of the quality metamodel. The organization of this quality model can be seen at (SQuaRE, 2005) specification. The Quality Model Class represents the quality attributes selected by the user to be maximized in the output model. A Quality model is composed main quality characteristics (functionality, reliability, usability, efficiency, maintainability and portability). Our quality model will have as the most six quality characteristics, depending on transformation's user selection. Each Quality Characteristic should be divided into sub-characteristics (this had also been defined in the ISO/IEC 9126-1). Finally each sub-characteristic can be sub divided in quality attributes, which will the ones that user selects for been maximized by the process.

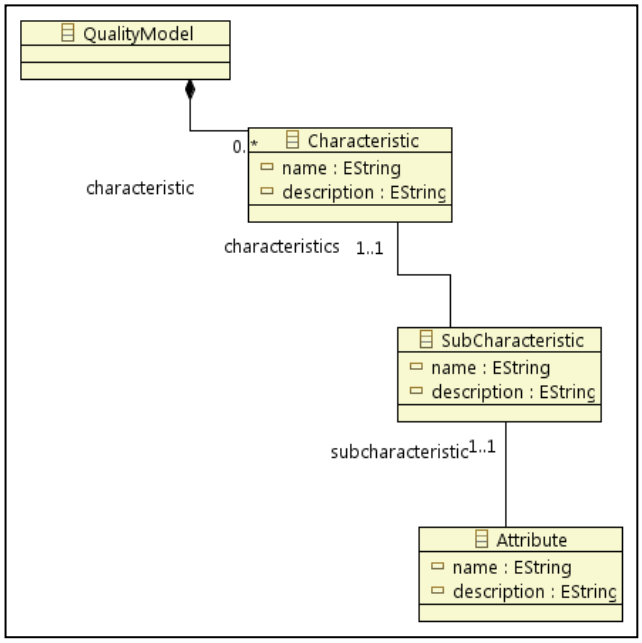


Illustration 1: Quality Metamodel

Transformation model: it will be needed a model to express the relationships between the alternative transformation rules that can be applied to a given set of constructs of the source model and the quality attributes that the selection of each alternative will give to the output model.

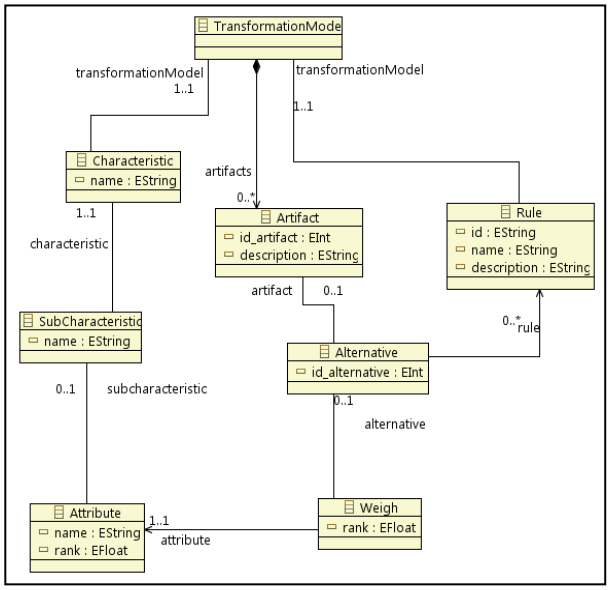


Illustration 10: Transformation Metamodel

Active Rules Set: this model will be the link result of the combination of the quality model and the transformation model. As explained before, the quality model exrpreses the quality attributes that the user of the transformation has selected to be maximized over the target model. In the first step of the transformation process The transformation model is explored searching for the alternatives that maximize the attributes selected by the user. For each estructural pattern an alternative is selected, and the rule set asociated to the alternative is instanciated in the Active Rules Model. A rule is define by a name and an Id. The Artifact is defined by its name.

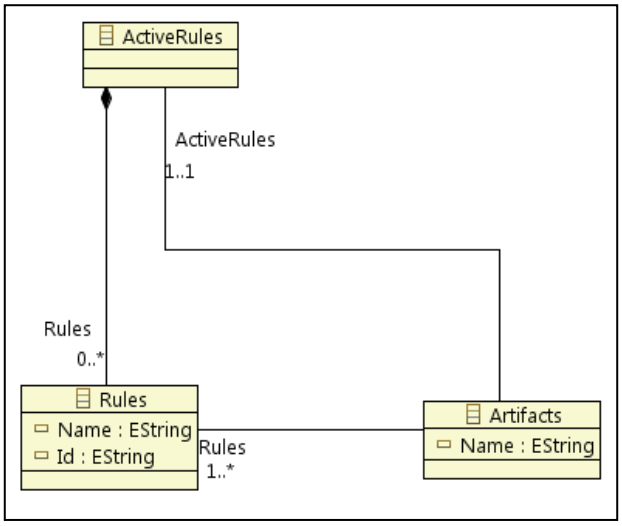


Illustration 11: Active Rules Metamodel

Active Rules Set Metamodel:

An Active Rules Set is composed have a list of artifacts (representing of set of constructs of the source model) and a set of rules that will be aplicable when this set of constructs is detected in the source model in the step 2 of the transformation process.

4.5 Strategy:

The Strategy of quality driven model transformations is based on the concept of making an assignment between patterns – pattern will be a set of source's metamodel entities as a portion of the source models- and quality attributes. We define the quality guidance when it's possible to have alternative transformations for a given source model's structural pattern. Those Alternative transformations finally consist of a set of rules which will be applicable in the transformation from the source to the target model.

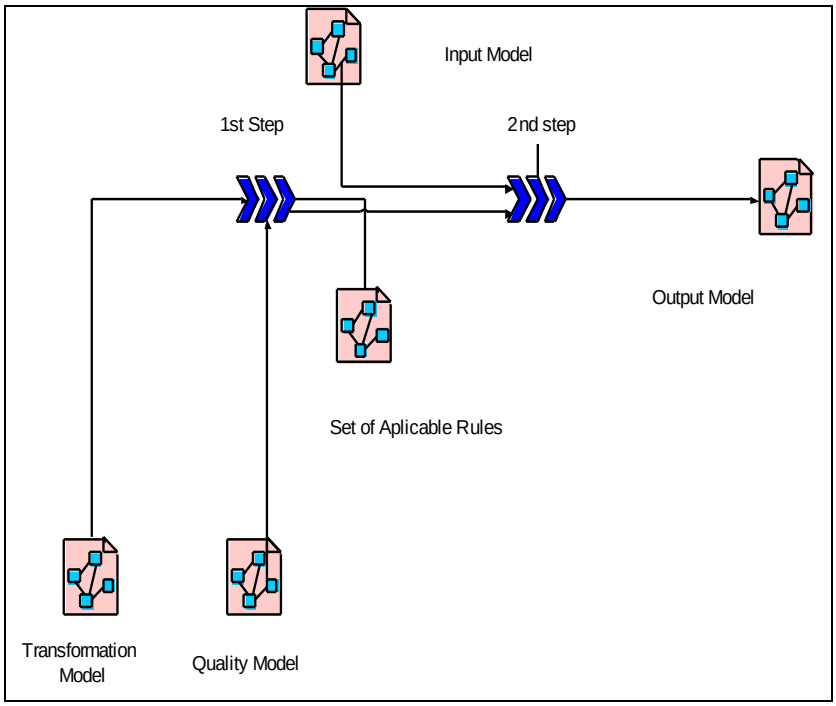


Illustration 2: Process Model

As seen in the section 4.2 the transformation process consists of two steps:

Step 1: Rule Analysis

With this first step we reach the domain independence. For any kind of source or target language, we can drive the transformation between them based on the inputs of the quality and transformation models and obtain a set of rules that will be fired when alternative transformations appear. Only rules related to quality attributes to be maximized will appear in the active rules set.

This phase is divided in two steps: i) Trade-off Analysis, ii) Automatic Rule Transformations.

Trade-off Analysis

The goal of this stage is to identify the design alternatives and their relative impact on the quality attributes. Specifically, we establish for each pattern in the source model the set of possible alternative rules. Moreover a set of quality attributes for each pattern is associated.

In order to solve the tradeoff between quality attributes, we apply the Analytic Hierarchy Process (AHP) (Saaty, 1990). AHP is an important decision making technique has been leveraged to resolve this kind of conflicts. AHP is based in rates for each alternative on each quality attribute. The assigned scores reflect the relative importance of the alternative.

The first step to apply the AHP is to build a graphical representation of the problem. The graphical is a hierarchy of the problem. The main goal is in the root. At the second level we put the quality attributes. This attributes contribute to achieve the main goal. At the second level

and we put the alternatives as leaf nodes. This decomposition allows decision makers to better understand every part of a complex problem.

The second step is to build the pair-wise comparisons matrix. For each quality attribute, the stakeholders provide the preferences on the alternatives comparing pairs of alternatives giving a weight in the AHP scale Table 3. This comparison is used to determine how important an A alternative, over a B alternative for a given quality attribute. We put the results in a comparison matrix and then we normalize. The score for each alternative is given by the average of each row in the normalized comparison matrix. A consistency check can be done to detect possible inconsistencies in the values.

Table 3 AHP weighting scale

If A is … as (than) B	Quantitative Weight
equally important	1
equality to moderate preferred	2
moderately more important	3
moderately to strong preferred	4
strongly more important	5
strongly to very strongly preferred	6
very strongly more important	7
very strongly to extremely preferred	8
extremely more important	9

The process done in the step 2 is repetead to elicit the importance for each quality attribute. The result es a comparation matrix with the relative importance of each quality attribute to the stakeholder.

Automatic Rule Transformations

At this stage we take as input two models: the quality model, and the transformation model. The **quality model** is composed by a hierarchy with three levels: characteristics, sub-characteristics and attributes. The **transformation model** contains for each artifact: the possible alternatives rules and the quality attributes to maximize for it. Based on the information extracted from the trade-off analysis, we include for each rule the relative weight for a given attribute, and the relative weight for each quality attributes.

These models are used to obtain an active rules set to apply. In order to obtain these rules, we apply a QVT transformation which chose the best rule for each artifact.

Step 2: Transformation

The second step is the original model transformation in the sense that takes the source model as input (and the *active rules set*) and converts it to the target model. In this step when a

structural pattern with alternative transformations is found in the source model the rules associated to this structural pattern in the *active rules set* will be applied.

The behavior on the second phase will be distinct if the set of constructs of the source model has or not alternative transformations. If the structural pattern have no alternative transformations the rule will perform the transformation with no references to the active rules set, these transformations are performed deterministically in the sense that are performed every time the estructural pattern is found in the source model (a source math has been found).

On the other hand we have the estructural patterns that had alternative transformations. When one estructural patterns is found on the source model only the rules that match with the active rules set for this pattern will be fired.

The transformation rule for these artifacts will be decomposed into two kinds of rules. The first will be auto-fired rule and the second will be a set of rules called from the first.

The auto-fired rule has two left sides, a left side for the source model domain in which the structural pattern of the source model is defined, and a left side for the active rules domain. The second left side domain checks that the artifact with the given name will be transformed by means of rules that will be called from this rule.

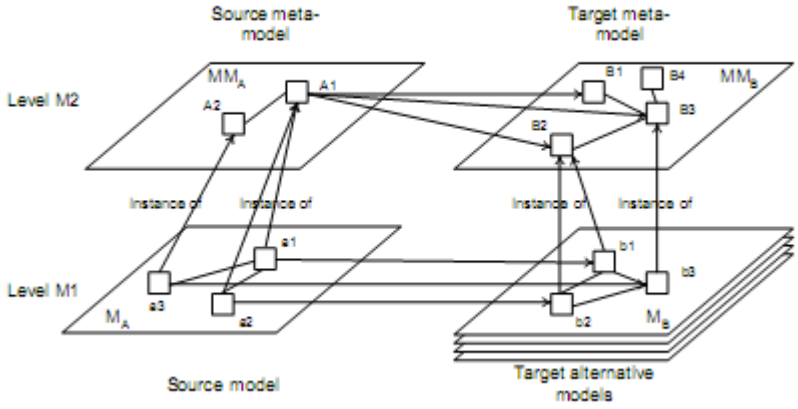


Illustration 13: different alternative transformations

I.E: Let's suppose we have a rule transforming Source "A1" into Target "B1". And lets suppose that "A1" can also be transformed into "B2". We will define in the transformation model two rules A1ToB1 and A1ToB2 and also we define the artifact A1 with two alternatives called A1B1 and A1B2. The A1B1 alternative is associated to the rule A1ToB1 and the A1B2 alternative is associated to the rule A1ToB2. When we execute the first step of the transformation rule we will have an active rule set containing the artifact "A1" with the rule A1ToB1 or the rule A1ToB2 but never both (those rules never will appear simultaneously in the active rule set).The definition of rule A1ToB1 will be formed by the domain that searches the "A1" in the source model and by a domain that searches for the name "A1ToB1" in the active rule set. There will be another domain for "B1" .

When a auto-fired rule has been fired having found a source match, then will call to the rules which will perform the transformation. The auto-fired rule will call to the entire set of rules associated to the structural pattern in the active-rules set.

The set of rules forming the different alternatives should cover the same portion of the input model. This concept is asociated to the completeness of the transformation. If one alterative cover less portion of the source model than the others when this alternative is selected a portion of the source model will not have correspondance in the target model.

4.6 Users of the transformation process

This architecture has three kinds of users or actors, the MDA User, The MDA Expert and the Domain Expert. The Domain Expert will define the quality attributes that will be applicable for the domain, and will assign weights to each attribute. Those weights will be used in a trade off process which populates the weights for the associations between quality attributes and the different alternative transformations. The User of the transformation typically will select the quality attributes in the quality model and will start the second phase of the transformation with a given source model. The MDA Expert will define the transformations, identifying the structural patterns of the source model and its equivalents in the target model. He also builds the transformation model, identifying alternative transformations for a given structural pattern. MDA Expert maintains the different artifacts and takes care of the result of the whole process.

Finally the domain expert will evaluate the whole process. This evaluation can be made applying different metrics to the output models. He can measure different quality attributes over the target model and make comparisons with the expected values. Those metrics may differ of the expected values. In this case the Domain Expert should review the association weights for alternatives and quality attributes. If those weights are incorrect the decision of what alternative should be also incorrect and perhaps this quality attributes aren't present in the output model. The process of correcting the weights for a given association between quality attributes and alternative transformations can be seen as a feedback for correcting the deviation between real metrics and expected values for those metrics.

The decision of what metrics should be applied for measuring a given quality attributes on a given model is taken by a MDA-Quality Expert, which may (or not) be the same than MDA-Expert.

4.7 Granularity of the transformation alternatives

At the architecture definition we had modeled the quality guidance defining a set of alternatives for transforming a set of constructs of the source model into constructs into the target model. Also we had considered the fact that the different alternatives must cover the same portion of the input model. In case of alternative transformations, this can be performed

by one or more rules (an alternative has a set of rules). This relaxes the restriction mentioned above, about the source model coverage.

But at this point we can see another issue, which is the granularity between estructural patterns. We define Granularity of a transformation as the size of the source model portion covered by the left side of the rule. A rule has high granularity if covers a wide structural pattern in the source model and another has low granularity if it covers a little number of constructs in the source model.

The left side of a rule also includes a checking for assuring that this artifact is not contained by a bigger artifact. The transformation model allows a given structural pattern been transformed by means of a set of rules. If a domain expert had defined that a big artifact should be transformed by means of a rule set and this alternative have been selected then this transformation should be applied as far as possible. It is possible for the domain expert to define big patterns been transformed into a complex pattern in the target model or into a wide set of simpler patterns, based on quality improvement of different attributes. If a given structural pattern in the source model is included into a bigger structural pattern the source match must be done with the biggest artifact. Once the MDA expert had defined a big structural pattern this must be taken into account as first priority (we make the supposition that a Domain Expert define high granularity patterns with the evidence or expecting that those transformations will improve the quality of the output model).

5. Application to an Specific Domain

This chapter shows how the architecture is applied to a specific domain: transformations between Requirements Model Sequence Diagrams to UML Class diagrams. The source and target models are presented and the different alternative transformations are defined.

5.1 Introduction

In this work we have chosen the transformations from Requirements model to UML Class Diagrams which is the same used in (Abrahão, 2008). We base our decision in two reasons, first is that both source and target domains are well known and the examples can be easy to understand and the traceability in the transformation process is easy to read and the second is that using the same specific domain than the work that described the problem will allow us to test our proposal and at the end we will be able to generalize our results.

5.2 The requirements model

The requirements model (Insfran, 2003) (E. Insfran, 2002) defines the structures and the process followed to capture the software requirements. It is composed of a *Functions Refinement Tree (FRT)* to specify the hierarchical decomposition of the system, a *Use Case Model* to specify the system communication and functionality, and *Sequence Diagrams* to specify the required object-interactions that are necessary to realize each Use Case. Consequently, as only functional software requirements are gathered (business requirements are excluded), the Requirements Model can be placed at the PIM level. The Requirements Model is supported by a Requirements Engineering Tool[2] (RETO).

Following a MDA strategy of model transformation, once the Requirements Model has been specified, a conceptual model including a UML class diagram can be obtained by applying a set of transformation rules from a Transformation Rules Catalog[3] (Insfran, 2003). These transformations establish traceability relationships between the Requirements Model and the UML class diagrams.

[2] RETO web site: http://reto.dsic.upv.es
[3] The Transformation Rules Catalog can be found in http://www.dsic.upv.es/~einsfran/thesis

According to the MOF terminology, the Requirements Model and the UML class diagram are located in the M1 level and their metamodels are located in the M2 level. The definition of a transformation is performed at the M2 level and implies that "*a certain structural pattern is identified in the source model (Requirements model), which corresponds to a valid structure in the target model (UML class diagram)*".

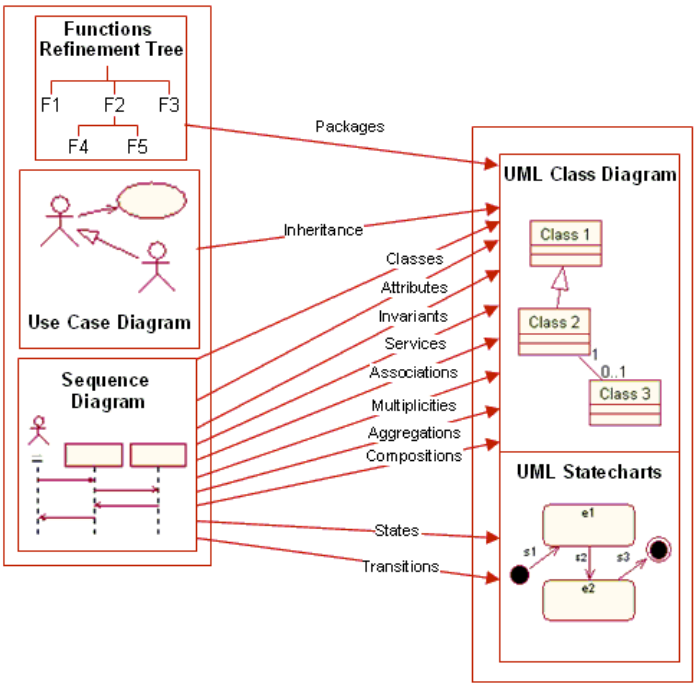


Illustration 14 Traceability from Requirements to Conceptual Models

The figure above shows simplified traceability relationship map to go from the set of specified requirements to specific elements in the conceptual schema. These traceability relationships may be simple (*one-to-one* relationships).

Traceability relationships can also be many-to-many relationships. This is due to the variability of the transformations, which allows multiple possible representations in the UML Class diagram that satisfy a given requirement pattern identified in the Requirements Model. When this occurs, this pattern will be represented by an artifact instance, and all the possible representations will be represented as alternatives with the related rules which will be applicable when the pattern is detected in the source model. Finally, single alternative mapping will be selected, based on the rules associated to the pattern on the Active Rules Set.

5.3 Requirements Metamodel

The Illustration 15shows an excerpt of the relevant parts of the Requirements Metamodel used as source in the transformation process. The Use Case class represents the functions of the system. Each Use Case will be specified in detail by means of one or more sequence diagrams. A Sequence Diagram is formed by Entities (Actor, Interfaces or classes) and Messages. The Actor represents de user of the system; Interface represents the system boundary; Class represents the different entity classes that participle in the realization of the Use Case.

In order to characterize the interaction between objects, we identify four types of messages: *Signal, Service, Query, and Connect*. Signal Messages represent the interaction between actors and the interface. Service messages represent object interactions with the purpose of modifying the system (creation, deletion or update). Query messages represent object interactions to query the state of an object or a set of objects. Connect messages represent object interactions to establish a relationship between them.

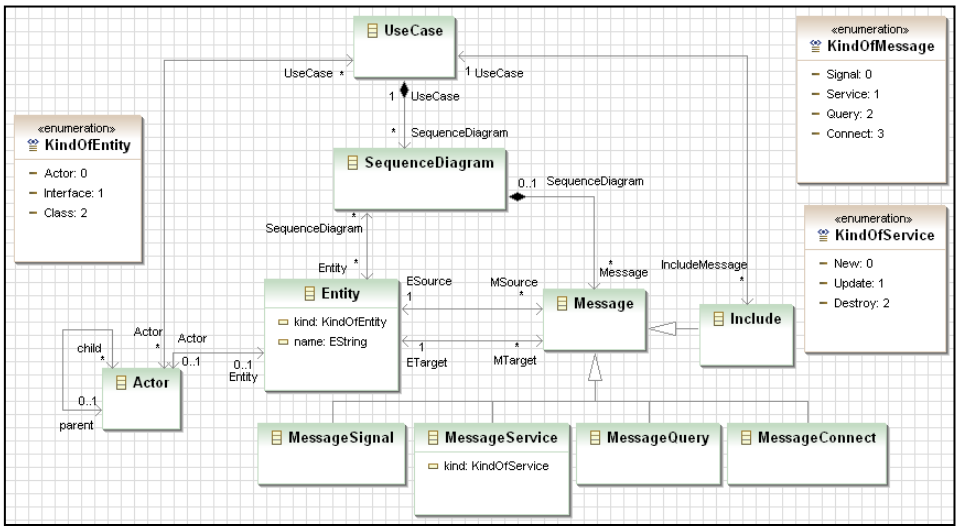


Illustration 15 Requirements Metamodel

5.4 The UML Class Diagram Metamodel

Once the source metamodel is defined, the UML target metamodel (OMG, 2006) must also be defined. At least three alternatives are possible:

- To use the UML2 metamodel directly. This has the advantage that the result can be used by all the tools that use this metamodel. However, the problem is the size of the metamodel and its complexity. The use of the UML2 metamodel makes transformation rules difficult to specify and understand.
- To use the Ecore metamodel. This has the advantage that many tools directly use this metamodel, and it is also very well integrated in the Eclipse environment (www.eclipse.org). However, we could not represent two of the three types of

relationships (*association class* and *aggregation*) that we needed to generate in this metamodel.

- To use the class diagram metamodel defined in the MOF QVT Final Adopted Specification (OMG, 2005). This metamodel is well known, simple, and it has the advantage that it can specify almost all the characteristics that are needed.

Finally, we decided to use a modified version of the class diagram of the MOF QVT specification, which we refer to as UMLite.

Illustration 16shows the modified UMLite metamodel. The main part of the metamodel is the same as the metamodel defined in the QVT specification (OMG, 2008) A *Package* is formed by a set of *PackageElements*. Usually, an information system is formed by a set of *Packages*. A *PackageElement* can be a *Classifier* or a *Relationship*. *Classifier* is the generic name given to everything that can have attributes and operations. *PrimitiveDataTypes* and *Classes* are both *Classifiers*. The class *PrimitiveDataType* defines the Abstract Data Types used in the definition of a system. Typical *PrimitiveDataTypes* are integers, doubles, strings, and so on. Instances of the *Class* class will belong to a specific *Package*. A *Class* is formed by a set of *Attributes*. Each one of the *Attributes* has a name inherited from *UMLModelElement* (in fact, everything has a name because every class inherits from the *UMLModelElement* class) and its type must be a *Classifier* that was previously defined. The IS-A relationship between classes is maintained with the reflexive association relationship defined in the *Class* class. The *Relationship* class defines the relationships that can exist between two classes (the source and the destination classes).

In order to be able to define the characteristics of relationships between classes, two modifications have been added to the metamodel:

- An attribute named *kind* in the *Relationship* class to express the kind of relationship between two classes (association, aggregation, or composition).
- A new relationship between the *Relationship* and *Class* classes to express that a relationship has an association class.

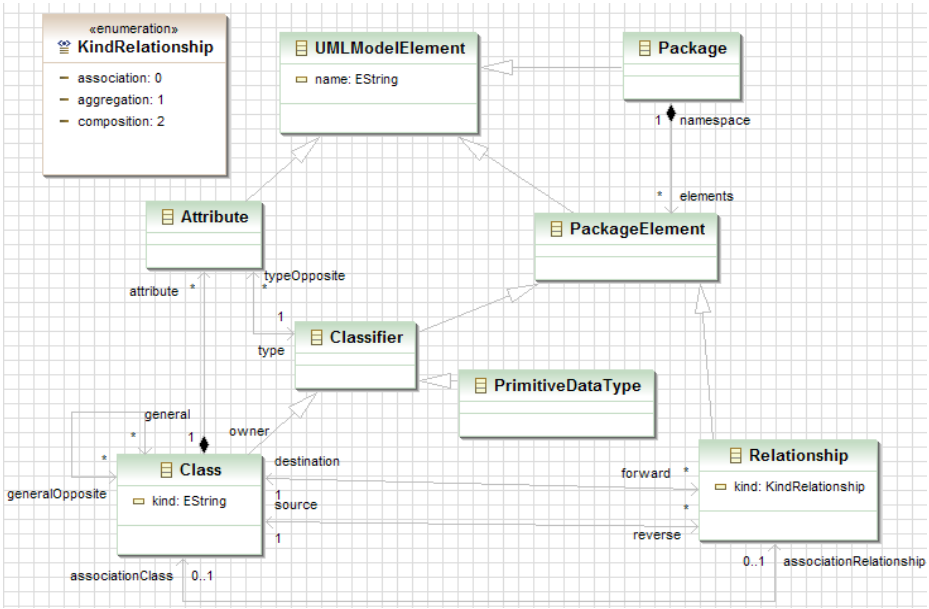


Illustration 16 The UMLite Metamodel

Even though there are no tools that use UMLite as their metamodel, it is still useful. Since the main concepts of UMLite are almost the same as the concepts in Ecore and UML2, they can be easily transformed to these metamodels.

5.5 Defining transformations:

5.5.1 Non Alternative Transformations:

The generation of classes for the UML class diagram is a process that is based on the analysis of participating actors and classes in all the Sequence Diagrams. It includes the application of the following Transformation Rules (TR), stated here in natural language:

- **TR 1.** For every distinct actor class participating in any Sequence Diagram, a class will be generated in the UML class diagram.
- **TR 2.** For every distinct class participating in any Sequence Diagram, a class will be generated in the UML class diagram.
- **TR 3.** The boundary classes (usually called Interface or System) in Sequence Diagrams will not have an explicit representation in the UML class diagram.

Those transformations are fired with no alternatives. For example the Rule 2 (TR2.) will be applied deterministically for every *distinct* class on the source model, and will transform this class into a UML class *when* the sequence diagram containing the class have been previously transformed.

This transformation rules will not be present in the transformation model nor in the active rules set. The non-alternative transformation are applied every time the structural pattern is found in the source model and without performing validations against the active rules set.

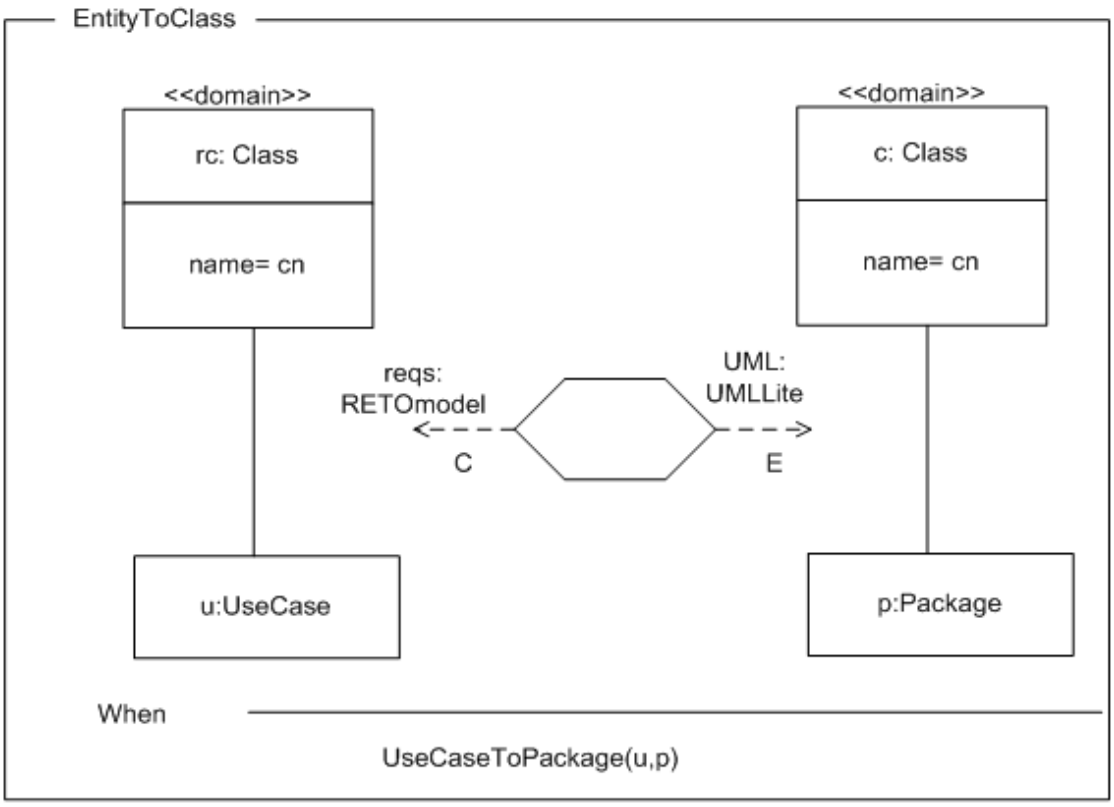


Illustration 17: Rule 2: Requirements class to UML Class in QVT Graphical Notation

The Illustration 17 shows the QVT graphical notation for the rule 2: Entity to class. The left side of the rule, the requirements domain, with the requirements class inside a sequence diagram and the right side, with the UMLite domain, and the UML Class inside a Package. The requirements domain has been defined as Checkonly and the UML has been defined as Enforce.

*reqs-uml.qvt

```
/* (JAV) Transformación de Entitys en Classes                                    */
/*       Busca en el modelo origen un UseCase y crea un paquete con el mismo nombre en el modelo destion*/
top relation EntityToClass
{
        an: String;
        /* (JAV) Busqueda de ClassSD (Clases del sequence diagram)*/
        checkonly domain reqs rc: metaRETOv01::sequenceDiagram::ClassSD
        {
            sequence_diagram=s: metaRETOv01::sequenceDiagram::SequenceDiagram
            {
            /*(JAV) Asegura que la clase este dentro del UseCase, luego con ello exigira que ese UseCase*/
            /*Haya sido transformado en un Package e incluira la clase que crea dentro del paquete*/
            useCases= u:metaRETOv01::useCaseDiagram::UseCase{}
            },
            name=an
        };
        /*(JAV) Creacion de la Clase*/
        enforce domain uml c: MyUMLite::Class  {
            namespace= p: MyUMLite::Package{},
            name = an
        };

        when {
        /* (JAV) Precondicion que el Caso de Uso se haya se haya convertido en Paquete*/
        UseCaseToPackage(u,p);
        }
}
```

Illustration 18: Rule 2: Requirements class to UML Class

5.5.2 Alternative Transformations:

On the other hand we have the alternative transformation. In (Abrahão, 2008) the authors had defined sets of alternatives (based on the output construct). Each rule is defined in terms of the source model structural patterns and the target construct that are generated when the rule is fired.

Table 4 Alternative transformations based on the output construct

Alternatives	Transformation Rules
A1 (association)	**TR 14.** For every message between two classes labeled with the stereotype «service/new» where both classes are distinct from the "interface" class THEN an association relationship between these classes will be generated. **TR 15.** For every message between two classes labeled with the stereotype «connect», THEN an association relationship between these classes will be generated. **TR 16.** For every message with the stereotype «service/new» or «connect» where classes using role names appears THEN an association relationship between these classes will be generated using these role names on the ends of the relationship.
A2 (aggregation)	**TR 28.** For every message with the stereotype «service/new» between two classes A and B, which are distinct from the "interface" class, THEN an aggregation relationship between these classes will be generated.
A3 (association class)	**TR 39.** For every message with the stereotype «service/new» from the class A to the class B, if there exist two messages with the stereotype «connect» starting from the class B to the classes C and D respectively, THEN a new association class will be generated (called B) AND also an association relationship between C and D related to the new association class B will be generated. **TR 40.** For every message with the stereotype «connect» from the class A to the class B, if there exist a message with the stereotype «service/new» starting from the class A or B to the class C THEN a new association class will be generated (called C) AND also, an association relationship between A and B related to the new association class C will be generated..

Those transformation rules are defined in Natural Language. If we translate this transformation rules into QVT we'll have alternative transformations, but at the moment we haven't quality guidance.

In order to achieve the quality guidance we should only apply the rule when has been selected in the first step of the process. In the definition shown before we must add the guidance in the way:

TR 15. For every message between two classes labeled with the stereotype «connect» in the source model, **when it had be selected the Rule TR15 between the alternative transformations for this constructs** THEN an association relationship between these classes will be generated.

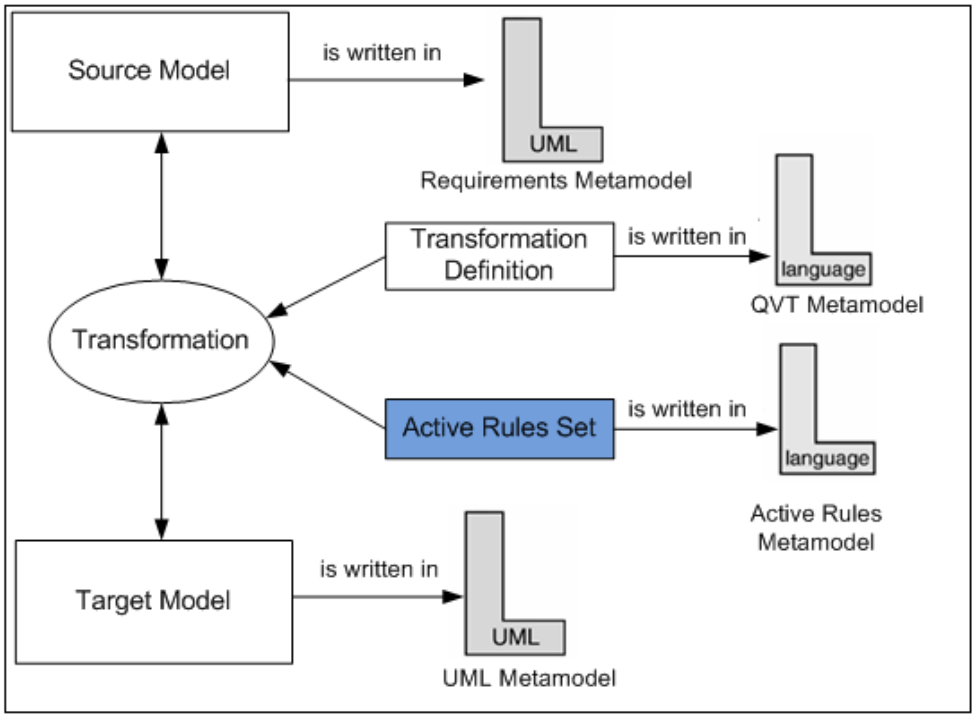


Illustration 19 Addition of the active rules set to the rule definition

As we define in section 5.4 there are two kind of rules, auto-fired rules (Top Relations in QVT Relations Syntax) and the rules invoked from the auto-fired (Relations in QVT Relations Syntax). These rules are fired each time that the Top Relation is executed without fails. The auto-fired rules consists of two domains in the left side, the first one is a structural pattern defined over the Requirements Metamodel, the second is defined over the *Active Rules Set* and searches for an artifact with the name given to those structural pattern. If both left sides are satisfied then a structural pattern will be created in the target model.

In the Illustration 20we can see QVT graphical syntax for the Connection rule, the structural pattern definition in the requirements metamodel: a connection between two classes contained in a sequence diagram called *nameAEntity* and *nameBEntity*. The match with the active rules set is expressed by means of a structural pattern defined over the active rules set metamodel. This definition consists of an artifact named “Connection” and will be associated with one rule. The structural pattern of the left side could have been included into rule with higher granularity. This is controlled with queries that assure that this structural pattern doesn’t form part of a bigger structural pattern. In the Example shown in the Illustration 10 the query appears in the when clause. Illustration 11 shows the QVT syntax of that query.

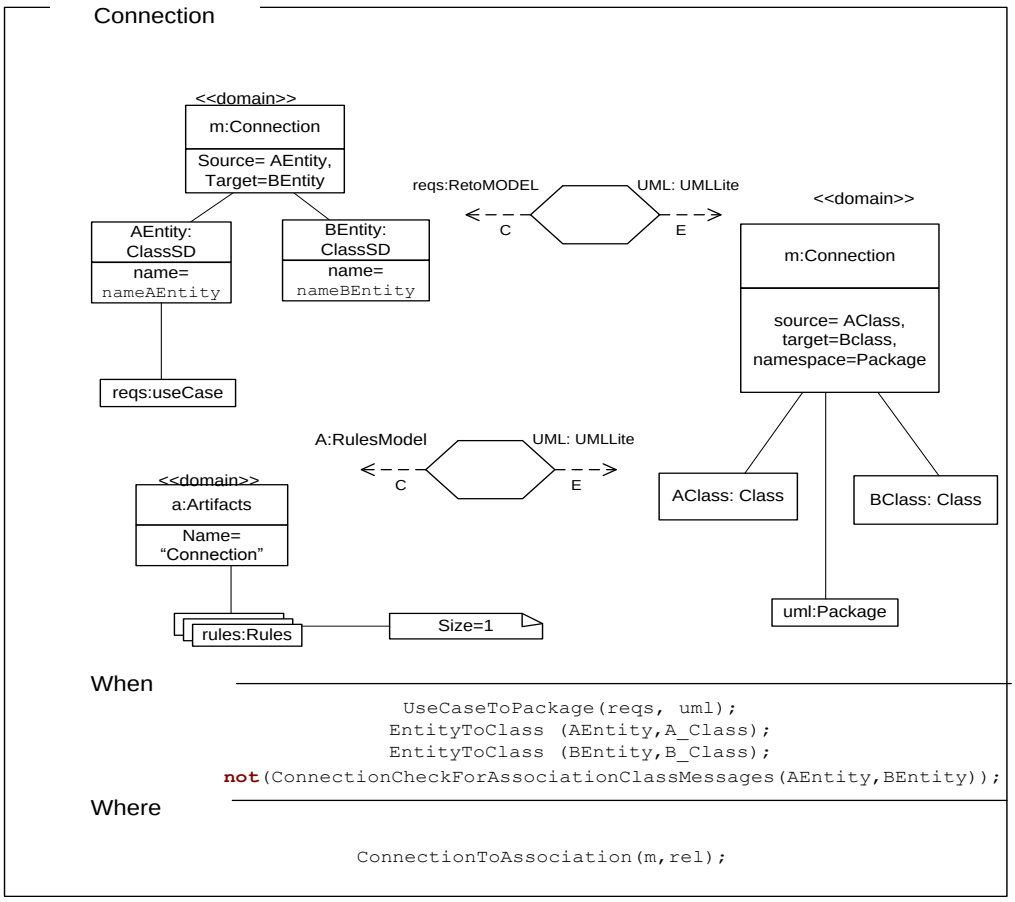


Illustration 20: Connection Rule in QVT Graphical Notation

The syntax of the verification queries is shown in the figure below. For a given two classes connected by a connection message it performs a verification in order to see it those classes are included in bigger structural patterns. For make this verification we see if there are other classes connected with service new messages that can be interpreted as association classes.

```
*reqs-uml.qvt
/* query: verifies if a Connection Pattern is included into a bigger Patter (AssociationClass)*/
query ConnectionCheckForAssociationClassMessages(AEntity: metaRETOv01::sequenceDiagram::ClassSD, BEntity: metaRETOv01::sequenceDiagram::ClassSD):Boolean
{

if (
    (
        (AEntity.sourceMessage->exists(m: metaRETOv01::sequenceDiagram::Message|m.oclIsTypeOf(metaRETOv01::sequenceDiagram::Service)))
        or
        (BEntity.sourceMessage->exists(m: metaRETOv01::sequenceDiagram::Message|m.oclIsTypeOf(metaRETOv01::sequenceDiagram::Service)))
    )
   or
    (
        (AEntity.targetMessage->exists(m: metaRETOv01::sequenceDiagram::Message|m.oclIsTypeOf(metaRETOv01::sequenceDiagram::Service)))
        and
        (AEntity.sourceMessage->exists(m: metaRETOv01::sequenceDiagram::Message|m.oclIsTypeOf(metaRETOv01::sequenceDiagram::Connection )implies m.target<>BEntity ))
    )
   )


then
true
else
false
endif

}
```

Illustration 21: ConnectionCheckForAssociationClassMessages QVT Syntax

At this point we have source model's structural patterns which can be transformed by its associated rule or, if this structural pattern is included into a bigger one then the rules associated with the bigger pattern will perform the transformation. But in our domain we have also structural patterns which can be transformed into various kinds of target structural patterns. For Example the rule 14 and the rule 28 are applied over the same left side. One Service New message between two classes can be transformed into an association or into an aggregation. In this case we can define a unique *Top Relation* in QVT Syntax and two *Non Top Relations* which will be fired from the Top Relation. Each Non Top Relation will transform the source structural pattern into an association or an aggregation depending on the rules appearing in the Active Rules Set. The Top relation will be formed by a left side with two domains, the requirements domain in which we define the service new message between two classes and the Active Rules Set in which we define the artifact called "`ServiceNew`". When Clause of the top relation is like Connection's When Clause. The difference appears at Where clause. In the Where Clause of the Service New Rule there appears a conditional, if the rule appearing in the active rules set for the Service New Artifact has an Id equals to 14 then the service new will be transformed into an Association Relationship, and if the rule appearing in the active rules set for the Service New Artifact has an Id equals to 14 then the service new will be transformed into an Aggregation Relationship. Both transformations will be performed by two different *non top relations* invoked from the where clause.

The Illustration 22 shows QVT graphical notation for the rule *ServiceNew*. The conditional *non-top relation* invocations can be seen in the where clause.

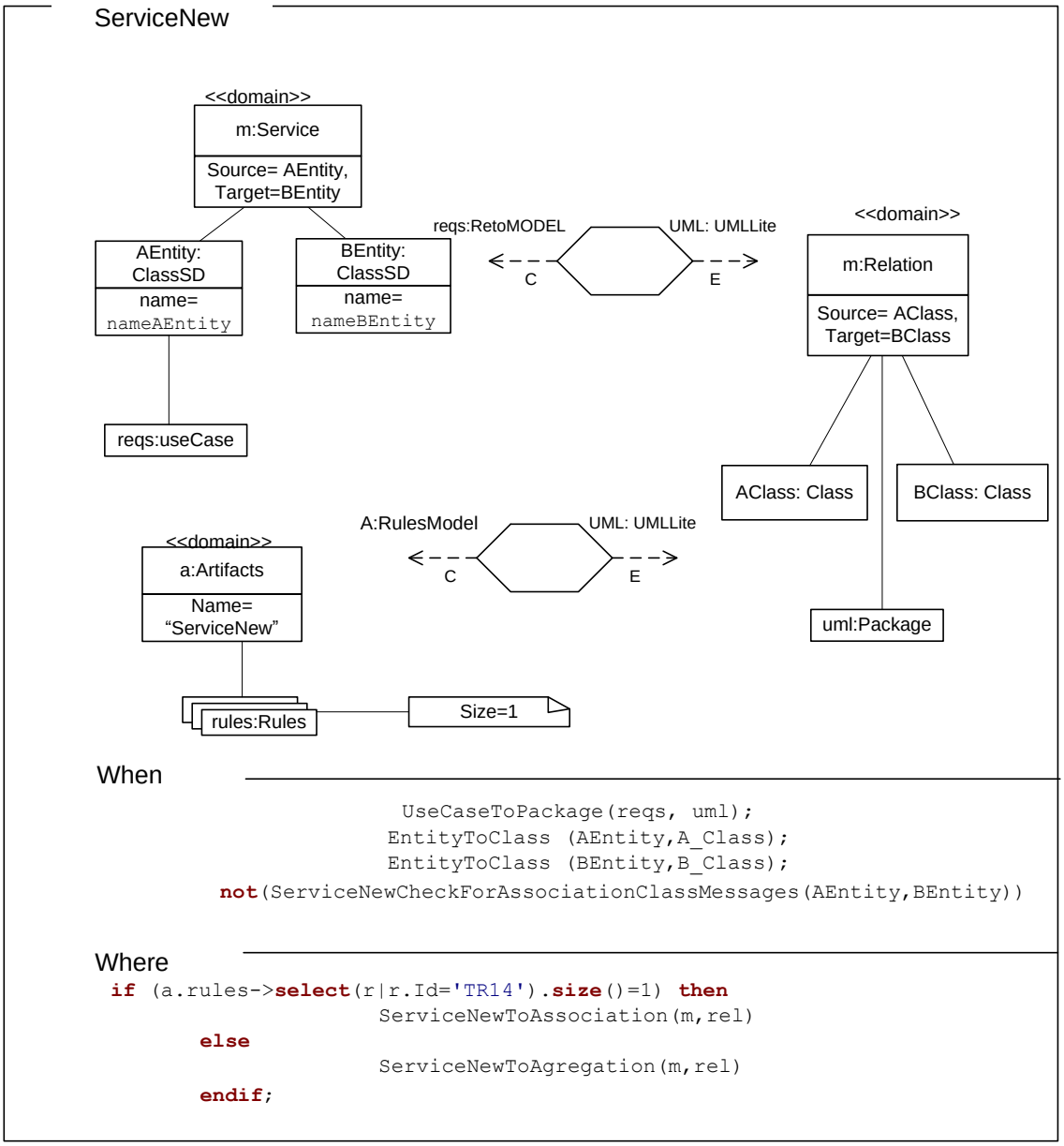


Illustration 22: Service-New Rule in QVT Graphical Notation

In our case, non top relations are characterized by left sides with a unique domain (the target domain), there's no need of active rules set references since those relations are directly (or recursively) invoked from the top relations where the references to the active rules set is defined. In our non top relations there´s no need of where clauses, these non top relations are defined in order to improve the reusability and the maintainability of the rules. Since there are many structural patterns which at last are transformed in combinations of little number of different elements (associations, aggregations, association-class etc) and the rules transforming into these elements can be reused, been invoked from the top level rules.

Illustration 23 shows the QVT graphical notation for the Service-New to Aggregation Rule, the Left side with the requirements domain, the right side of the rule with the UML domain and the where clause in which previous transformations are defined. These non top relations finally perform the transformation from the requirements metamodel pattern into the UML Class diagrams metamodel in the alternative transformations.

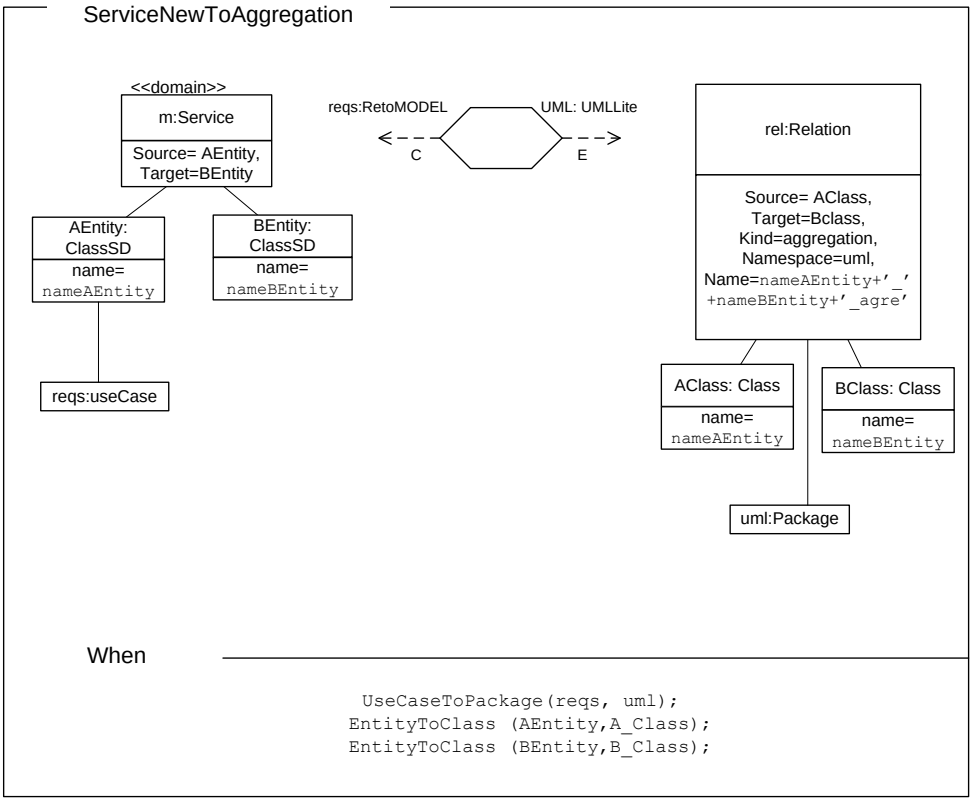


Illustration 23: Service New To aggregation Rule in QVT Graphical Notation

The rule transforming a Service New Message into a Association will be very similar to the rule shown in the Illustration 23. The main difference between ServiceNewToAggregation and ServiceNewToAsociation is the enforce domain, the kind of the relation which is created changes from aggregation to association.

Both the ServiceNewToAggregation and the ServiceNewToAggregation, and others as ConnectToAssociation can be called from higher granularity rule. This bigger pattern rules can transform the structural source pattern into a structural pattern with high granularity or into a set of low granularity structural patterns. Those low granularity structural patterns can be associations or aggregations.

For example the Structural Source Pattern defined in the left side of the rule 40 can be transformed into an Association-Class (as defined in the rule 40) but also can be transformed into two associations or an association and an aggregation.

If the domain expert decides to define bigger source structural patterns, then the target pattern will be combination of the rules presented at this point as set of rules associated to the source structural pattern in the transformation model at the step 1.

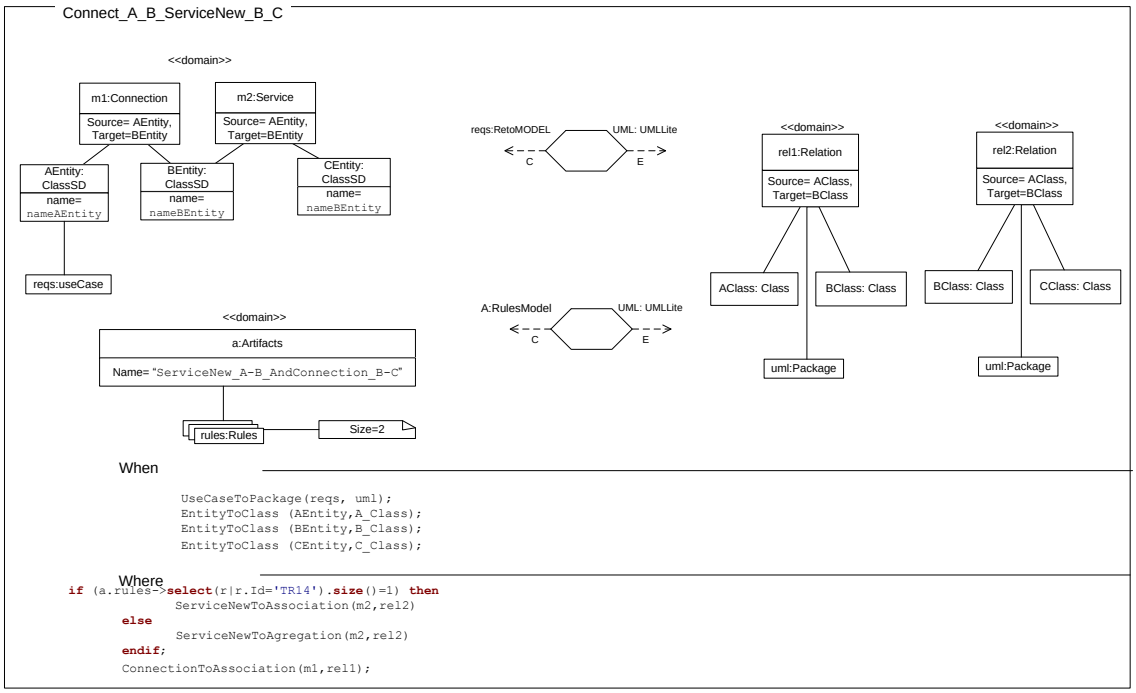


Illustration 24: Connection A B and Service New B C in QVT Graphical notation

The Illustration 24 shows the rule for transforming a structural pattern with the granularity of an association-class into two low-granularity structural patterns (can be two associations or an association plus an aggregation). This rule is fired whenever the active rules set has two rules for the structural pattern “Connection_A-B_ServiceNewAnd_B-C” and this structural pattern is in the source model (without being included into bigger structural patterns).

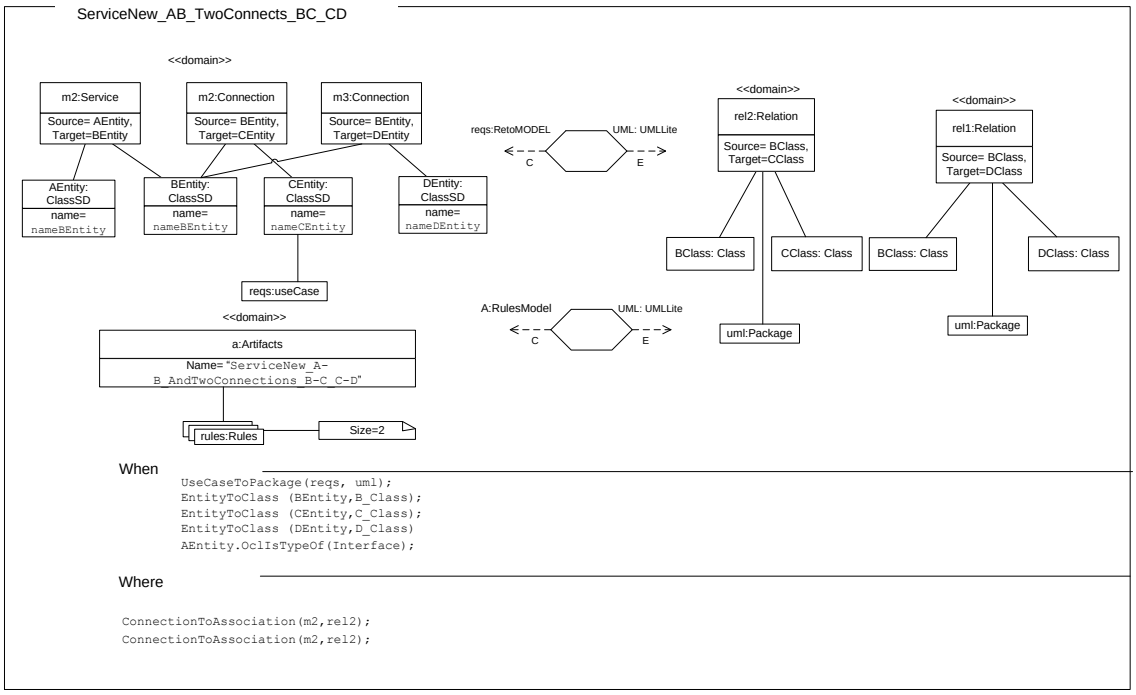


Illustration 25: Service New A B and Two connections B C and BD in QVT Graphical notation

Illustration 25 shows the rule for transforming a structural pattern with the granularity of an association-class into two low-granularity structural patterns. This rule is fired whenever the active rules set has two rules for the structural pattern “ServiceNew_A-B_AndTwoConnects_B-C” and this structural pattern is in the source model (without being included into bigger structural patterns).

6. Proof of Concept

This chapter shows how the alternatives are applied in a transformation process and how a sequence diagram generates different UML class diagrams. In order to illustrate the transformation process and the alternative transformations we had used a example taken from the specification of a Hotel Management System.

6.1 Introduction

To specify the object interactions to realize the Use Case *Room Rental* that is initiated by the the employee actor we had used a Sequence Diagram. This Use Case represents the rent of a room by a customer.

Illustration 26 shows the sequence diagram for the *Room Rental* use case. First the actor starts the process, the system ask the actor for the User's Id. Once the user has entered the required data then system performs a searching for the system. Once the customer has been retrieved, ask for the rental data (start date and end date). The system then performs a search for free rooms. If there are free rooms in the hotel in the given dates then the user selects one and creates the Rental. The Rental object will be then associated with the customer and the room. Finally the system creates the cost of this room (the invoicing data for the room) and associates this cost with the rental. The Illustration 27 shows an excerpt for the representation of this sequence diagram according with the RETO Metamodel. Messages which generates no transformations have been excluded from this representation in order to clearify the models.

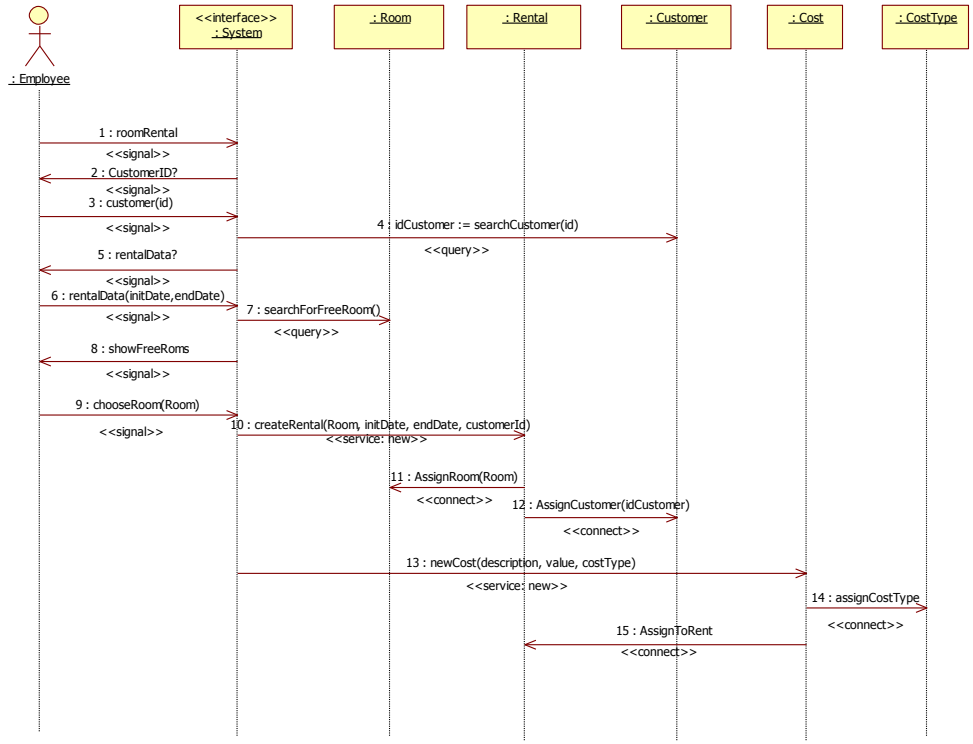


Illustration 26: Interactions for the Use Case Room Rental

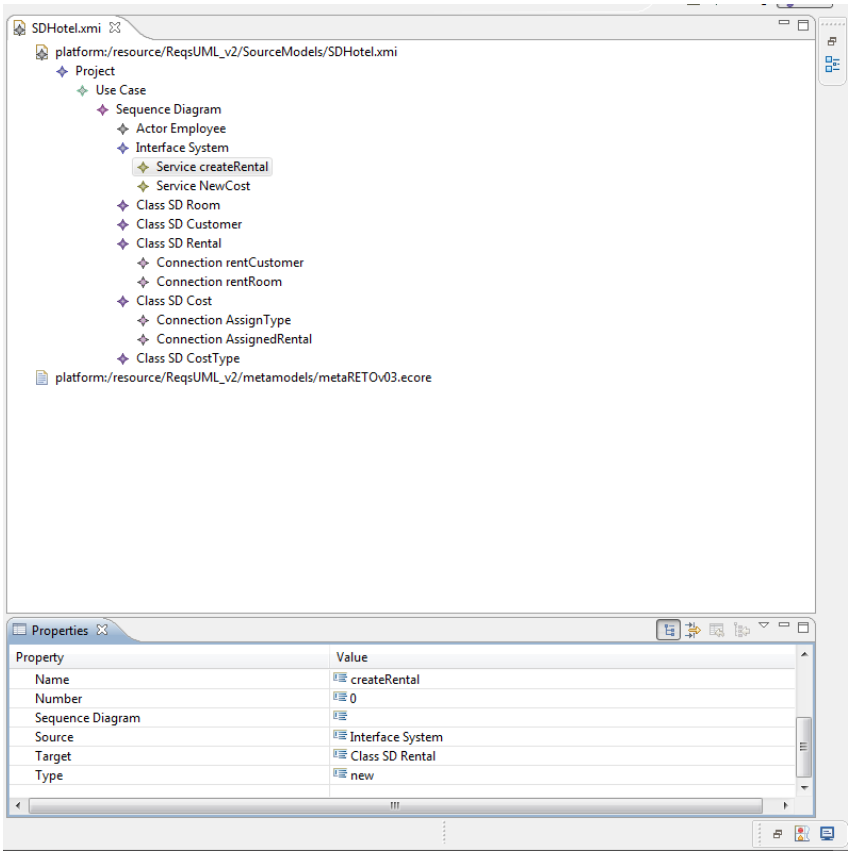
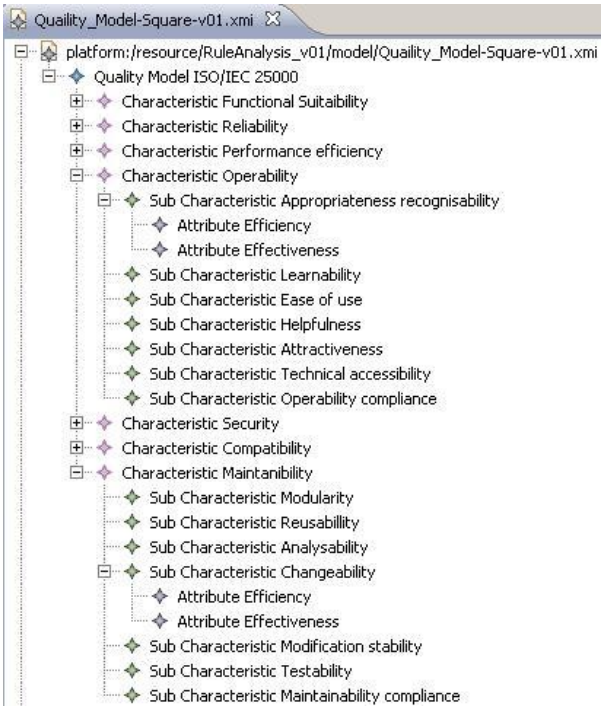


Illustration 27: Ecore representation of the Room Rental Sequence Diagram defined according with RETO Metamodel

6.2 Step 1: Rule Analysis

In our approach this information, together with the transformation and the quality models will be used to transform the requirements specification into a UML class diagram. There are alternative transformations which can be applied, and as result of the model transformation process different output models could be generated.

Our Quality Model is based on the Square standard [(SQuaRE, 2005)]. For this domain, we choose two characteristics to compare: Operability (referred as Usability in the ISO 9126-1) and Maintainability. We made this selection; due the fact the Operability is one of the main factors influencing the Maintainability.



Applying the AHP technique, the expert domain decides the rates for the quality attributes and the alternatives for each quality attribute. There are two rather different interpretations of how the MDA vision might be achieved. These two schools of thought have been termed “elaborationist” and “translationist” (McNeile, 2003) In the elaborationist approach, the definition of the application is built up gradually as you progress through from PIM to PSM to Code. In the other hand, in the translationist approach the PIM is translated directly into the final code of the system by code generation. Clearly, in the elaborationist approach, the generated artefacts (the PSM and the Code) must be understandable to the developer; otherwise modification (elaboration) would be not being possible. For our case study we use an elaborationist approach, due the fact that the target model is not directly transformed to a code model. For this reason we assign the weights in the way stated above (see table 4).

Table 5 Quality Model Weights

Quality Attribute	Weight
Appropriateness Recognisability Efficiency	0,40
Appropriateness Recognisability Effectiveness	0,35
Changeability Efficiency	0,15
Changeability Effectiveness	0,10

The second step in the AHP application is to give weights to the alternatives. In our case study we consider two alternatives to the structural pattern: "*For every message with the stereotype «service/new» from the class A to the class B, and two messages with the stereotype «connect» starting from the class B to the classes C and D respectively*". In the first, the whole source pattern is transformed into two associations (the A entity is an interface and then the service new will not be transformed) by a call to the **rule TR15**. The alternative is that the whole source pattern is transformed into a new association class (called B) AND also an association relationship between C and D related to the new association class B will be generated with a call to the **rule TR39**.

Table 6: Alternative transformations based on the output construct

Alternatives	Transformation Rules
A1 (association)	**TR 14.** For every message between two classes labeled with the stereotype «service/new» where both classes are distinct from the "interface" class THEN an association relationship between these classes will be generated. **TR 15.** For every message between two classes labeled with the stereotype «connect», THEN an association relationship between these classes will be generated. **TR 16.** For every message with the stereotype «service/new» or «connect» where classes using role names appears THEN an association relationship between these classes will be generated using these role names on the ends of the relationship.
A2 (aggregation)	**TR 28.** For every message with the stereotype «service/new» between two classes A and B, which are distinct from the "interface" class, THEN an aggregation relationship between these classes will be generated.
A3 (association class)	**TR 39.** For every message with the stereotype «service/new» from the class A to the class B, if there exist two messages with the stereotype «connect» starting from the class B to the classes C and D respectively, THEN a new association class will be generated (called B) AND also an association relationship between C and D related to the new association class B will be generated. **TR 40.** For every message with the stereotype «connect» from the class A to the class B, if there exist a message with the stereotype «service/new» starting from the class A or B to the class C THEN a new association class will be generated (called C) AND also, an association relationship between A and B related to the new association class C will be generated..

The alternative weights were given by the domain experts based in two empirical experiments. In the first work by Abrahão et al. (Abrahão, 2008), is presented a controlled experiment to determine which of the transformation rules for structural relationships: association (A1), aggregation (A2), or association class (A3), obtained the easiest to understand UML class model. The results show a slight tendency to favor the transformations related to associations (A1). Based in this study we can conclude that the diagrams generated applying the rule **TR15** has a better Appropriateness Recognisability over the rule **TR39**. The second work (Marcela Genero, 2007) demonstrates with an empirical experiment that the Number of Associations (NAssoc) of one Class Diagram is related with the Maintainability. This fact implies that the TR15 generated model has a worst Maintainability compared with the TR39 generated one. The table 3 show the weights assigned to the alternatives according with the empirical experiments.

Table 7 Rule Weights

Rule	Appropriateness Recognisability		Changeability	
	Efficiency	Effectiveness	Efficiency	Effectiveness
TR15	0,40	0,35	0,25	0,14
TR39	0,36	0,30	0,40	0,37

6.3 Step 2: Model Transformation

As an alternative solution to the transformation process based on the entities and messages contained on the sequence diagram shown on the Illustration 26 and together with transformation and quality models two possible output models are shown on Table 86. Those output models have been generated applying the transformations defined in section 5.5.1 and 5.5.2.

The alternative output class models shown in the Table 8 can be obtained by applying the alternative transformations associated to the structural pattern "ServiceNewAB_Connects_BC_BD" who has been matched twice. In the first match the messages involved are 10 (Service), 11 and 12 (Connects). In the second match the messages involved are 13 (Service), 14 and 15 (Connects). It is obvious that other low granularity patterns can be recognized in the sequence diagram. Although in section 4.7 we establish that the match must be against the high granularity rule in order to avoid inconsistencies.

Once the source models structural patterns and the matches have been defined, the fired rules will depend on the selection made by the rule analysis. With the rules defined at this moment we have two alternatives:

- The whole source pattern is transformed into two associations (the A entity is an interface and then the service new will not be transformed) by a call to the rule 15
- The whole source pattern is transformed into a new association class (called B) AND also an association relationship between C and D related to the new association class B will be generated with a call to the rule 39.

If the selection in the rule analysis was (a) the message 11 will be transformed into an association to relate the Room rented with the Rental. The message 12 will be transform into an association to relate the Customer with the Rental. In the same way the message 14 will became an association to relate Cost with the Rental and the 15 will became an association to relate Cost with the CostType.

If the selection in the rule analysis was (b) the messages 10, 11 and 12 will be transformed into an association class (named Rental) to relate the Room rented with the Customer. Moreover the messages 13, 14 and 15 will be transformed into an as an association class (named Cost) to relate the CostType with the Rental.

If the attributes to be maximized includes *Appropriateness Recognisability Efficiency* or *Appropriateness Recognisability Effectiveness* then the model generated is shown in the illustration 8a with a NAssoc value of 4. If the attributes to be maximized includes *Changeability Effectiveness* or *Changeability Efficiency* the model generated is shown in the illustration 8.b with a NAssoc value of 2.

As was defined in section 6.2 the metric Nassoc in Class Diagram is related with the Maintainability. The output model when *Changeability Effectiveness* or *Changeability Efficiency* is selected has better values for this metrics and the presence of Association classes makes the models more difficult to understand. The output model when *Appropriateness Recognisability Efficiency* or *Appropriateness Recognisability* is easier to understand based on the presence of associations. However the output model has a worse value for NAssoc Metrics as a consequence has worse Maintainability.

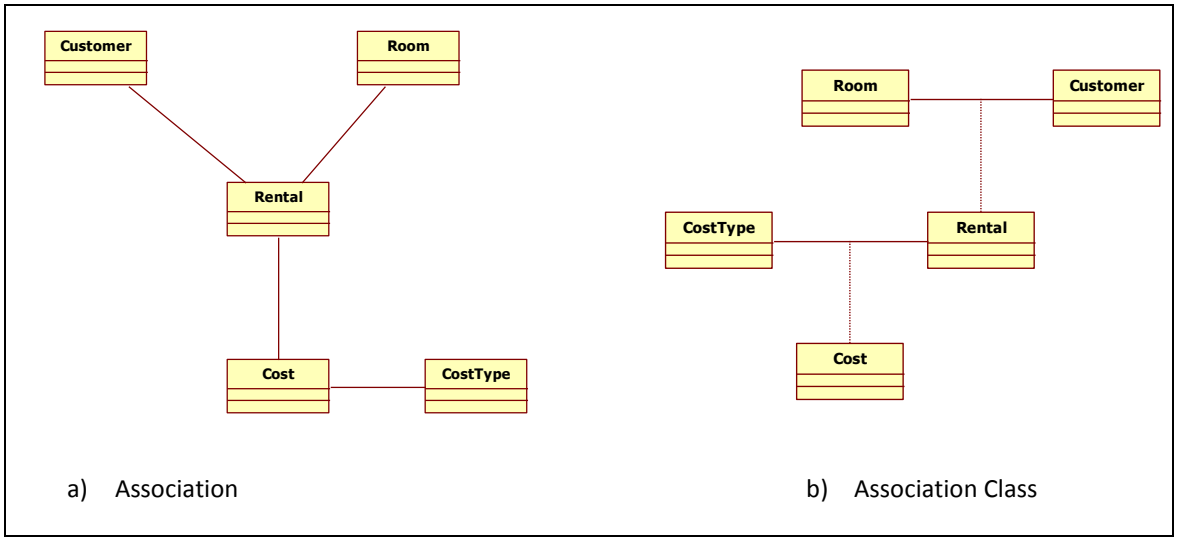


Table 8 two alternative class models for the Room Rental System

6.4 Conclusions

In this chapter, we have presented a proof of concept based on a requirement specification of a system and using an instance of the architecture we can see the impact of the selection of a quality attribute has on the output model. In addition, we show that the output model satisfies the selected quality attributes. This not only evidences the right behavior of the transformation architecture, but also that the relationships among quality attributes and alternative transformations were correctly defined.

7. Conclusions

In this chapter, we will explain the contributions, future works related with this final project and the results obtained in the context of this project.

7.1 Conclusions and Future Work

We started the project with the (Abrahão, 2008) proposal and the state of art in quality driven transformations. Based on transformations defined in (Insfran, 2003) we defined the first approach of the architecture.

The main contribution of this project is the definition of a generic architecture to perform quality-driven model transformations. This architecture allows the definition of a quality model, a transformation model, and active rules model as artifacts to be used in the quality-driven model transformation process. Our strategy was based on divide the process in two steps. First consists in the selection of the alternatives which will be applied for each structural pattern. This selection is made based on the weights of the different alternative transformations given by the domain expert and the relative relevance and the attributes selected by the user. The second is the application of the selected alternatives to the source model of the transformation.

All the artifacts needed for the transformations where defined in Eclipse Development Environment, first the metamodels were designed and then all the models were defined and tested. The transformation rules were defined using Medini QVT and tested with many examples to show the feasibility of the approach.

As future work, we plan to perform a complete case study to test and improve the proposed architecture. In addition, we plan to study the inter-relationship among quality attributes and to perform a trade-off analysis to identify the design alternatives and their relative impact on the quality attributes.

7.2 Results obtained

The results of this project have been sent to the 11th International Conference on Product Focused Software Development and Process Improvement (PROFES 2010), June 21-23, Limerick, Ireland http://www.lero.ie/profes2010/

Appendix A: Model Transformations

```
transformation reqs_uml(reqs:metaRETOv01, uml: MyUMLite, rul:
MyRulesModel){
/* (JAV) Transforamcion de UseCase a Package
                                                        */
/*          Busca en el modelo origen un UseCase y crea un paquete con
el mismo nombre en el modelo destion*/
/* Prueba */
key MyUMLite::Relationship {source, destination, asociation_class };
key MyUMLite::Class {name,namespace};



top relation UseCaseToPackage
{
            ucn: String;
            /*(JAV) Use case en el modelo origen*/
            checkonly            domain            reqs            r:
metaRETOv01::useCaseDiagram::UseCase
            {
                  name = ucn
            };
            /*(JAV) Creacion del Package*/
            enforce domain  uml u :MyUMLite::Package
            {
                  Name=ucn
            };
}
/* (JAV) Transformación de Entitys en Classes
                                                              */
/*          Busca en el modelo origen un UseCase y crea un paquete con
el mismo nombre en el modelo destion*/
top relation EntityToClass
{
            an: String;
            /*  (JAV)  Busqueda  de  ClassSD  (Clases  del  sequence
diagram)*/
            checkonly            domain            reqs            rc:
metaRETOv01::sequenceDiagram::ClassSD
            {
                  sequence_diagram=s:
metaRETOv01::sequenceDiagram::SequenceDiagram
                  {
                  /*(JAV) Asegura que la clase este dentro del UseCase,
luego con ello exigira que ese UseCase*/
                  /*Haya sido transformado en un Package e incluira la
clase que crea dentro del paquete*/
                  useCases= u:metaRETOv01::useCaseDiagram::UseCase{}
                  },
                  name=an
            };
```

```
            /*(JAV) Creacion de la Clase*/
            enforce domain uml c: MyUMLite::Class  {
                  namespace= p: MyUMLite::Package{},
                  name = an
            };

            when {
            /* (JAV) Precondicion que el Caso de Uso se haya se haya
convertido en Paquete*/
            UseCaseToPackage(u,p);
            }
}



/*TR40 MessageConnect And Service::New -> Asociation Class*/
top relation Connection_A_B_AndServiceNew_A_C
{
      nameAEntity, nameBEntity, nameCEntity: String;
            checkonly                  domain                  reqs
m1:metaRETOv01::sequenceDiagram::Connection
            {

                  source=                                    AEntity:
metaRETOv01::sequenceDiagram::ClassSD
                  {
                  name=nameAEntity,
                  sequence_diagram=s:
metaRETOv01::sequenceDiagram::SequenceDiagram
                        {
                        /*el UseCase es comprobado contra la clausula
del when*/
                        useCases=
reqs:metaRETOv01::useCaseDiagram::UseCase{}
                        }
                  ,

      sourceMessage=m2:metaRETOv01::sequenceDiagram::Service
                  {
                        target=                              CEntity:
metaRETOv01::sequenceDiagram::ClassSD
                        {
                              name=nameCEntity
                        }
                  }
                  },
                  target=BEntity: metaRETOv01::sequenceDiagram::ClassSD
                  {
                              name=nameBEntity
                  }



            };


            checkonly domain rul a: MyRulesModel::Artifacts
      {
            Name='ServiceNew_A-B_AndConnection_A-C'
```

```
      };

            enforce domain uml rel1: MyUMLite::Relationship{
                        source=A_Class:MyUMLite::Class {},
                        destination= B_Class:MyUMLite::Class{},
                        namespace= PaqUml: MyUMLite::Package{}


            };

            enforce domain uml rel2: MyUMLite::Relationship{
                        source=A_Class:MyUMLite::Class {},
                        destination= C_Class:MyUMLite::Class{},
                        namespace= PaqUml: MyUMLite::Package{}


            };
when
{
            m2.id_message=m1.id_message+1;
            a.rules->size()=2;
            UseCaseToPackage(reqs, PaqUml);
            EntityToClass (AEntity,A_Class);
            EntityToClass (BEntity,B_Class);
            EntityToClass (CEntity,C_Class);

}
where
{

      if          (a.rules->select(r|r.Id='TR14').size()=1)          then
ServiceNewToAssociation(m2,rel2)
      else
            ServiceNewToAgregation(m2,rel2)
      endif;
      ConnectionToAssociation(m1,rel1);


}
}




top relation ServiceNew_A_B_AndConnection_A_C_AsocClass
{
      nameAEntity, nameBEntity, nameCEntity: String;
            checkonly                        domain                        reqs
m1:metaRETOv01::sequenceDiagram::Connection
            {

                  source=                                                AEntity:
metaRETOv01::sequenceDiagram::ClassSD
                  {
                  name=nameAEntity,
                  sequence_diagram=s:
metaRETOv01::sequenceDiagram::SequenceDiagram
                        {
```

```
                        /*el UseCase es comprobado contra la clausula
del when*/
                        useCases=
reqs:metaRETOv01::useCaseDiagram::UseCase{}
                        }
                  ,

      sourceMessage=m2:metaRETOv01::sequenceDiagram::Service
                  {
                        target=                                    CEntity:
metaRETOv01::sequenceDiagram::ClassSD
                        {
                              name=nameCEntity
                        }
                  }
                  },
                  target=BEntity: metaRETOv01::sequenceDiagram::ClassSD
                  {
                              name=nameBEntity
                  }



            };


            checkonly domain rul a: MyRulesModel::Artifacts
      {
            Name='ServiceNew_A-B_AndConnection_A-C'


      };



            enforce domain uml rel: MyUMLite::Relationship{

                        namespace= PaqUml: MyUMLite::Package{}


            };
when
{
            m2.id_message=m1.id_message+1;
            a.rules->size()=1;
            a.rules->select(r|r.Id='TR40').size()=1;
            UseCaseToPackage(reqs, PaqUml);

}
where
{

      ConnectionAndServiceNewToAssociationClass(m1,m2,rel);

}
}

top relation ServiceNewAndTwoConnects_AsocClass {

            id_mess1, id_mess2, id_mess3: Integer;
```

```
            checkonly                    domain                    reqs
m1:metaRETOv01::sequenceDiagram::Service {
                  id_message=id_mess1,
                  source= AEntity: metaRETOv01::elements::Entity
                  {

                        sequence_diagram=s:
metaRETOv01::sequenceDiagram::SequenceDiagram
                        {
                        /*el UseCase es comprobado contra la clausula
del when*/
                        useCases=
reqs:metaRETOv01::useCaseDiagram::UseCase{}
                        }
                  },

                  target=BEntity: metaRETOv01::sequenceDiagram::ClassSD
                  {

                  sourceMessage=                                    m2:
metaRETOv01::metaRETOv01::sequenceDiagram::Connection
                  {
                        id_message=id_mess2,

                        target                              =CEntity:
metaRETOv01::sequenceDiagram::ClassSD
                        {

                        }
                  }
                  }



            };
            checkonly          domain          reqs          m3:
metaRETOv01::metaRETOv01::sequenceDiagram::Connection
                  {
                        id_message=id_mess3,
                        target                              =DEntity:
metaRETOv01::sequenceDiagram::ClassSD
                        {

                        }
                  };


            enforce domain uml rel: MyUMLite::Relationship {

                        asociation_class= B_Class: MyUMLite::Class{},
                        source=C_Class:MyUMLite::Class{},
                        destination= D_Class:MyUMLite::Class {},
                        kind = KindOfRelationShip :: asociation,


                  namespace= PaqUml: MyUMLite::Package
                        {
                        }
            };

            checkonly domain rul a: MyRulesModel::Artifacts {
```

```
                Name='ServiceNew_A-B_And2Connection_B-C_C-D'
            };

            when {

            EntityToClass (BEntity,B_Class);
            EntityToClass (CEntity,C_Class);
            EntityToClass (DEntity,D_Class);


            m3.source=BEntity;
            m2<>m3;
            id_mess1<id_mess3;
            id_mess2<id_mess3;
            id_mess3<id_mess1+3;


            a.rules->size()=1;
            a.rules->select(r|r.Id='TR39').size()=1;
            CEntity<>DEntity;

      (AEntity.oclIsTypeOf(metaRETOv01::useCaseDiagram::Interface))  or
(AEntity.oclIsTypeOf(metaRETOv01::sequenceDiagram::ClassSD) );
            UseCaseToPackage(reqs, PaqUml);
            }

            where {
            ServiceAndTwoConnectionsToAssociationClass(m1,m2,m3,rel);
            }

      }

top relation ServiceNewAndTwoConnectsNoInterface {

            id_mess1, id_mess2, id_mess3: Integer;
            checkonly                    domain                    reqs
m1:metaRETOv01::sequenceDiagram::Service {
                  id_message=id_mess1,
                  source= AEntity: metaRETOv01::elements::Entity
                  {

                        sequence_diagram=s:
metaRETOv01::sequenceDiagram::SequenceDiagram
                        {
                        /*el UseCase es comprobado contra la clausula
del when*/
                        useCases=
reqs:metaRETOv01::useCaseDiagram::UseCase{}
                        }
                  },

                  target=BEntity: metaRETOv01::sequenceDiagram::ClassSD
                  {

                  sourceMessage=                                    m2:
metaRETOv01::metaRETOv01::sequenceDiagram::Connection
                  {

                        id_message=id_mess2,
                        target                              =CEntity:
metaRETOv01::sequenceDiagram::ClassSD
```

```
                        {

                        }
                  }
                  }


            };
            checkonly            domain            reqs            m3:
metaRETOv01::metaRETOv01::sequenceDiagram::Connection
                  {

                        id_message=id_mess3,
                        target                                    =DEntity:
metaRETOv01::sequenceDiagram::ClassSD
                        {

                        }
                  };


            enforce domain uml rel1: MyUMLite::Relationship {
                  source=A_Class:MyUMLite::Class{},
                  destination=B_Class:MyUMLite::Class{},
                  namespace= PaqUml: MyUMLite::Package
                        {
                        }
            };

            enforce domain uml rel2: MyUMLite::Relationship {
                  source=B_Class:MyUMLite::Class{},
                  destination=C_Class:MyUMLite::Class{},
                  namespace= PaqUml: MyUMLite::Package
                        {
                        }
            };
            enforce domain uml rel3: MyUMLite::Relationship {
                  source=C_Class:MyUMLite::Class{},
                  destination=D_Class:MyUMLite::Class{},
                  namespace= PaqUml: MyUMLite::Package
                        {
                        }
            };


            checkonly domain rul a: MyRulesModel::Artifacts {
                  Name='ServiceNew_A-B_And2Connection_B-C_C-D'
            };

            when {



            EntityToClass (AEntity,A_Class);
            EntityToClass (BEntity,B_Class);
            EntityToClass (CEntity,C_Class);
            EntityToClass (DEntity,D_Class);


            m3.source=BEntity;
```

```
            m2<>m3;
            id_mess1<id_mess3;
            id_mess2<id_mess3;
            id_mess3<id_mess1+3;



            a.rules->size()=2;
            CEntity<>DEntity;
            (AEntity.oclIsTypeOf(metaRETOv01::sequenceDiagram::ClassSD)
);
            UseCaseToPackage(reqs, PaqUml);
            }

            where {

                  if  (a.rules->select(r|r.Id='TR14').size()=1) then
                        ServiceNewToAssociation(m1,rel1)
                  else
                        ServiceNewToAgregation(m1,rel1)
                  endif;
            ConnectionToAssociation(m2,rel2);
            ConnectionToAssociation(m3,rel3);

            }

      }

top relation ServiceNewAndTwoConnectsInterface {

            id_mess1, id_mess2, id_mess3: Integer;
            checkonly                  domain                        reqs
m1:metaRETOv01::sequenceDiagram::Service {
                  id_message=id_mess1,
                  source= AEntity: metaRETOv01::elements::Entity
                  {

                        sequence_diagram=s:
metaRETOv01::sequenceDiagram::SequenceDiagram
                        {
                        /*el UseCase es comprobado contra la clausula
del when*/
                        useCases=
reqs:metaRETOv01::useCaseDiagram::UseCase{}
                        }
                  },

                  target=BEntity: metaRETOv01::sequenceDiagram::ClassSD
                  {

                  sourceMessage=                                      m2:
metaRETOv01::metaRETOv01::sequenceDiagram::Connection
                  {
                        id_message=id_mess2,
                        target                                  =CEntity:
metaRETOv01::sequenceDiagram::ClassSD
                        {

                        }
                  }
```

```
                  }


            };
            checkonly            domain            reqs            m3:
metaRETOv01::metaRETOv01::sequenceDiagram::Connection
                  {
                        id_message=id_mess3,
                        target                                    =DEntity:
metaRETOv01::sequenceDiagram::ClassSD
                        {

                        }
                  };



            enforce domain uml rel2: MyUMLite::Relationship {
                  source=B_Class:MyUMLite::Class{},
                  destination=C_Class:MyUMLite::Class{},
                  namespace= PaqUml: MyUMLite::Package
                        {
                        }
            };
            enforce domain uml rel3: MyUMLite::Relationship {
                  source=C_Class:MyUMLite::Class{},
                  destination=D_Class:MyUMLite::Class{},
                  namespace= PaqUml: MyUMLite::Package
                        {
                        }
            };


            checkonly domain rul a: MyRulesModel::Artifacts {
                  Name='ServiceNew_A-B_And2Connection_B-C_C-D'
            };

            when {

            EntityToClass (BEntity,B_Class);
            EntityToClass (CEntity,C_Class);
            EntityToClass (DEntity,D_Class);


            m3.source=BEntity;
            m2<>m3;
            id_mess1<id_mess3;
            id_mess2<id_mess3;
            id_mess3<id_mess1+3;

            a.rules->size()=2;
            CEntity<>DEntity;

      (AEntity.oclIsTypeOf(metaRETOv01::useCaseDiagram::Interface));
            UseCaseToPackage(reqs, PaqUml);
            }

            where {
```

```
            ConnectionToAssociation(m2,rel2);
            ConnectionToAssociation(m3,rel3);

            }

      }


top relation Connection_A_B_AndServiceNew_B_C
{
      nameAEntity, nameBEntity, nameCEntity: String;
            checkonly                  domain                  reqs
m1:metaRETOv01::sequenceDiagram::Connection
            {

                  source=                                    AEntity:
metaRETOv01::sequenceDiagram::ClassSD
                  {
                  name=nameAEntity,
                  sequence_diagram=s:
metaRETOv01::sequenceDiagram::SequenceDiagram
                        {
                        /*el UseCase es comprobado contra la clausula
del when*/
                        useCases=
reqs:metaRETOv01::useCaseDiagram::UseCase{}
                        }

                  },
                  target=BEntity: metaRETOv01::sequenceDiagram::ClassSD
                  {
                              name=nameBEntity,

      sourceMessage=m2:metaRETOv01::sequenceDiagram::Service
                        {
                              target=                          CEntity:
metaRETOv01::sequenceDiagram::ClassSD
                              {
                                    name=nameCEntity
                              }
                        }

                  }



            };


            checkonly domain rul a: MyRulesModel::Artifacts
      {
            Name='ServiceNew_A-B_AndConnection_B-C'


      };

            enforce domain uml rel1: MyUMLite::Relationship{
                        source=A_Class:MyUMLite::Class {},
                        destination= B_Class:MyUMLite::Class{},
                        namespace= PaqUml: MyUMLite::Package{}
```

```
            };

            enforce domain uml rel2: MyUMLite::Relationship{
                        source=B_Class:MyUMLite::Class {},
                        destination= C_Class:MyUMLite::Class{},
                        namespace= PaqUml: MyUMLite::Package{}


            };
when
{

            a.rules->size()=2;
            UseCaseToPackage(reqs, PaqUml);
            EntityToClass (AEntity,A_Class);
            EntityToClass (BEntity,B_Class);
            EntityToClass (CEntity,C_Class);

}
where
{

      if          (a.rules->select(r|r.Id='TR14').size()=1)          then
ServiceNewToAssociation(m2,rel2)
      else
            ServiceNewToAgregation(m2,rel2)
      endif;
      ConnectionToAssociation(m1,rel1);


}
}




top relation ServiceNew
{
      nameAEntity, nameBEntity: String;
            checkonly                     domain                     reqs
m:metaRETOv01::sequenceDiagram::Service
            {

                  source=                                          AEntity:
metaRETOv01::sequenceDiagram::ClassSD
                  {
                  name=nameAEntity,
                  sequence_diagram=s:
metaRETOv01::sequenceDiagram::SequenceDiagram
                        {
                        /*el UseCase es comprobado contra la clausula
del when*/
                        useCases=
reqs:metaRETOv01::useCaseDiagram::UseCase{}
                        }

                  },
                  target=BEntity: metaRETOv01::sequenceDiagram::ClassSD
```

```
                  {
                              name=nameBEntity
                  }
                  };

                  checkonly domain rul a: MyRulesModel::Artifacts
            {
                  Name='ServiceNew'


            };

                  enforce domain uml rel: MyUMLite::Relationship{
                        source=A_Class:MyUMLite::Class {},
                        destination= B_Class:MyUMLite::Class{},
                        namespace= PaqUml: MyUMLite::Package{}


                  };
                  when
                  {
                        a.rules->size()=1;
                        UseCaseToPackage(reqs, PaqUml);
                        EntityToClass (AEntity,A_Class);
                        EntityToClass (BEntity,B_Class);

      not(ServiceNewCheckForAssociationClassMessages(AEntity,BEntity))
;
                  }
                  where
                  {
                  if    (a.rules->select(r|r.Id='TR14').size()=1)    then
ServiceNewToAssociation(m,rel)
                  else
                        ServiceNewToAgregation(m,rel)
                  endif;
                  }
}


top relation Connection
{
      nameAEntity, nameBEntity: String;
            checkonly                     domain                        reqs
m:metaRETOv01::sequenceDiagram::Connection
            {

                  /* Message's Source Entity*/
                  source=                                             AEntity:
metaRETOv01::sequenceDiagram::ClassSD
                  {
                  name=nameAEntity,
                  sequence_diagram=s:
metaRETOv01::sequenceDiagram::SequenceDiagram
                        {

                        useCases=
reqs:metaRETOv01::useCaseDiagram::UseCase{}
                        }

                  },
```

```
                  /* Message's Target Entity*/
                  target=BEntity: metaRETOv01::sequenceDiagram::ClassSD
                  {
                              name=nameBEntity
                  }
                  };
                  /* This rule must be the selected for the construct
set in the active rules set*/
                  checkonly domain rul a: MyRulesModel::Artifacts
            {
                  Name='Connection'


            };
                  /* Target Construct Set */
                  enforce domain uml rel: MyUMLite::Relationship{
                        source=A_Class:MyUMLite::Class {},
                        destination= B_Class:MyUMLite::Class{},
                        namespace= PaqUml: MyUMLite::Package{}


                  };
                  when
                  {
                        /* this only will be applicable whe the source
construct set is transformed with a single rule*/
                        a.rules->size()=1;
                        /* the Source's Use Case have been previously
transformed into a Package*/
                        UseCaseToPackage(reqs, PaqUml);
                        /* Entities of the source model have been
transformed into target's model classes*/
                        EntityToClass (AEntity,A_Class);
                        EntityToClass (BEntity,B_Class);
                        /* the message is'nt a part of a bigger
construct (it will be transformed using another rule*/

      not(ConnectionCheckForAssociationClassMessages(AEntity,BEntity,m
));
                  }
                  where
                  {
                        /*It will perform the transformation*/
                        ConnectionToAssociation(m,rel);

                  }
}

relation ConnectionToAssociation
{

            nameAEntity, nameBEntity: String;
            checkonly                  domain                  reqs
m1:metaRETOv01::sequenceDiagram::Connection
            {

                  source=                                    AEntity:
metaRETOv01::sequenceDiagram::ClassSD
                  {
                  name=nameAEntity,
```

```
                  sequence_diagram=s:
metaRETOv01::sequenceDiagram::SequenceDiagram
                        {
                        /*el UseCase es comprobado contra la clausula
del when*/
                        useCases=
reqs:metaRETOv01::useCaseDiagram::UseCase{}
                        }
                  }
                  ,

                  target=BEntity: metaRETOv01::sequenceDiagram::ClassSD
                  {
                              name=nameBEntity
                  }



            };



            enforce domain uml rel: MyUMLite::Relationship{
                        kind = KindOfRelationShip :: asociation,
                        name= nameAEntity + '_' + nameBEntity+'_asoc',
                        namespace= PaqUml: MyUMLite::Package{}


            };
            when
            {
                  UseCaseToPackage(reqs, PaqUml);
            }
}



relation ServiceNewToAssociation{

            nameAEntity, nameBEntity: String;
            checkonly                  domain                    reqs
m1:metaRETOv01::sequenceDiagram::Service
            {

                  source=                                    AEntity:
metaRETOv01::sequenceDiagram::ClassSD
                  {
                  name=nameAEntity,
                  sequence_diagram=s:
metaRETOv01::sequenceDiagram::SequenceDiagram
                        {
                        /*el UseCase es comprobado contra la clausula
del when*/
                        useCases=
reqs:metaRETOv01::useCaseDiagram::UseCase{}
                        }
                  }
                  ,

                  target=BEntity: metaRETOv01::sequenceDiagram::ClassSD
```

```
                {
                        name=nameBEntity
                }


        };


        enforce domain uml rel: MyUMLite::Relationship{
                source=A_Class:MyUMLite::Class {},
                destination= B_Class:MyUMLite::Class{},
                kind = KindOfRelationShip :: asociation,
                name= nameAEntity + '_' + nameBEntity+'_asoc',
                namespace= PaqUml: MyUMLite::Package{}


        };
        when
        {
        EntityToClass (AEntity,A_Class);
        EntityToClass (BEntity,B_Class);
        UseCaseToPackage(reqs, PaqUml);
        }
}




relation ServiceNewToAgregation{

        nameAEntity, nameBEntity: String;
        checkonly                domain                reqs
m1:metaRETOv01::sequenceDiagram::Service
        {

                source=                                AEntity:
metaRETOv01::sequenceDiagram::ClassSD
                {
                name=nameAEntity,
                sequence_diagram=s:
metaRETOv01::sequenceDiagram::SequenceDiagram
                        {
                        /*el UseCase es comprobado contra la clausula
del when*/
                        useCases=
reqs:metaRETOv01::useCaseDiagram::UseCase{}
                        }
                }
                ,

                target=BEntity: metaRETOv01::sequenceDiagram::ClassSD
                {
                        name=nameBEntity
                }
```

```
            };

            enforce domain uml rel: MyUMLite::Relationship{
                        source=A_Class:MyUMLite::Class {},
                        destination= B_Class:MyUMLite::Class{},
                        kind = KindOfRelationShip :: agregation,
                        name= nameAEntity + '_' + nameBEntity+'_agre',
                        namespace= PaqUml: MyUMLite::Package{}

            };
            when
            {
            EntityToClass (AEntity,A_Class);
            EntityToClass (BEntity,B_Class);
            UseCaseToPackage(reqs, PaqUml);
            }
}


relation ConnectionAndServiceNewToAssociationClass{

            nameAEntity, nameBEntity, nameCEntity: String;
            checkonly                    domain                    reqs
m1:metaRETOv01::sequenceDiagram::Connection
            {

                  source=                                          AEntity:
metaRETOv01::sequenceDiagram::ClassSD
                  {
                  name=nameAEntity,
                  sequence_diagram=s:
metaRETOv01::sequenceDiagram::SequenceDiagram
                        {
                        /*el UseCase es comprobado contra la clausula
del when*/
                        useCases=
reqs:metaRETOv01::useCaseDiagram::UseCase{}
                        }
                  }
                  ,

                  target=BEntity: metaRETOv01::sequenceDiagram::ClassSD
                  {
                              name=nameBEntity
                  }



            };

            checkonly                    domain                    reqs
m2:metaRETOv01::sequenceDiagram::Service
            {
```

```
                target=CEntity: metaRETOv01::sequenceDiagram::ClassSD
                {
                            name=nameCEntity
                }


        };


        enforce domain uml rel: MyUMLite::Relationship{
                        source=A_Class:MyUMLite::Class {},
                        destination= B_Class:MyUMLite::Class{},
                        asociation_class= C_Class: MyUMLite::Class{},
                        kind = KindOfRelationShip :: asociation,
                        name=       nameAEntity       +       '_'       +
nameBEntity+'_asoc_class_'+nameCEntity,
                        namespace= PaqUml: MyUMLite::Package{}


        };
        when
        {
        m2.source=BEntity or m2.source=AEntity;
        EntityToClass (AEntity,A_Class);
        EntityToClass (BEntity,B_Class);
        EntityToClass (CEntity,C_Class);
        UseCaseToPackage(reqs, PaqUml);


        }
}



relation ServiceAndTwoConnectionsToAssociationClass{


        nameBEntity, nameCEntity,nameDEntity: String;
        checkonly                   domain                      reqs
m1:metaRETOv01::sequenceDiagram::Service
        {

                source= AEntity:  metaRETOv01::elements::Entity
                {
                sequence_diagram=s:
metaRETOv01::sequenceDiagram::SequenceDiagram
                        {
                        /*el UseCase es comprobado contra la clausula
del when*/
                        useCases=
reqs:metaRETOv01::useCaseDiagram::UseCase{}
                        }
                }
                ,

                target=BEntity: metaRETOv01::sequenceDiagram::ClassSD
                {
```

```
                        name=nameBEntity
                }


        };


        checkonly                       domain                          reqs
m2:metaRETOv01::sequenceDiagram::Connection
        {


                target=CEntity: metaRETOv01::sequenceDiagram::ClassSD
                {
                        name=nameCEntity
                }


        };

        checkonly                       domain                          reqs
m3:metaRETOv01::sequenceDiagram::Connection
        {


                target=DEntity: metaRETOv01::sequenceDiagram::ClassSD
                {
                        name=nameDEntity
                }


        };

        enforce domain uml rel: MyUMLite::Relationship{
                        asociation_class= B_Class: MyUMLite::Class{},
                        source=C_Class:MyUMLite::Class{},
                        destination= D_Class:MyUMLite::Class {},
                        kind = KindOfRelationShip :: asociation,
                        name=       nameCEntity       +       '_'       +
nameDEntity+'_asoc_class_39'+nameBEntity,
                        namespace= PaqUml: MyUMLite::Package{}


        };
        when
        {

      (AEntity.oclIsTypeOf(metaRETOv01::useCaseDiagram::Interface))  or
(AEntity.oclIsTypeOf(metaRETOv01::sequenceDiagram::ClassSD) );
        m2<>m3;
```

```
            CEntity<>DEntity;
            EntityToClass (BEntity,B_Class);
            EntityToClass (CEntity,C_Class);
            EntityToClass (DEntity,D_Class);
            UseCaseToPackage(reqs, PaqUml);



            }
}


/*(JAV) funcion para comprobar los mensajes entrantes y salientes de
las entidades, para forzar que no puedan tener tipos mezclados*/
query               ServiceNewCheckForAssociationClassMessages(AEntity:
metaRETOv01::sequenceDiagram::ClassSD,                          BEntity:
metaRETOv01::sequenceDiagram::ClassSD)
:Boolean
{

if (
      (
            (AEntity.sourceMessage->exists(m:
metaRETOv01::sequenceDiagram::Message|m.oclIsTypeOf(metaRETOv01::seque
nceDiagram::Connection)))
            or
            (AEntity.targetMessage->exists(m:
metaRETOv01::sequenceDiagram::Message|m.oclIsTypeOf(metaRETOv01::seque
nceDiagram::Connection)))
      )
   or
      (BEntity.sourceMessage->select(m:
metaRETOv01::sequenceDiagram::Message|m.oclIsTypeOf(metaRETOv01::seque
nceDiagram::Connection) )->size()>=2)
   )
then
true
else
false
endif

}

/*(JAV) funcion para comprobar los mensajes entrantes y salientes de
las entidades, para forzar que no puedan tener tipos mezclados*/
query               ConnectionCheckForAssociationClassMessages(AEntity:
metaRETOv01::sequenceDiagram::ClassSD,                          BEntity:
metaRETOv01::sequenceDiagram::ClassSD,                             asoc:
metaRETOv01::sequenceDiagram::Connection ):Boolean
{

if (
      (
            (AEntity.sourceMessage->exists(m:
metaRETOv01::sequenceDiagram::Message|m.oclIsTypeOf(metaRETOv01::seque
nceDiagram::Service)))
            or
            (BEntity.sourceMessage->exists(m:
metaRETOv01::sequenceDiagram::Message|m.oclIsTypeOf(metaRETOv01::seque
nceDiagram::Service)))
```

```
      )
   or
      (
            (AEntity.targetMessage->exists(m:
metaRETOv01::sequenceDiagram::Message|m.oclIsTypeOf(metaRETOv01::seque
nceDiagram::Service) and (m.id_message < asoc.id_message)))
            and
            (AEntity.sourceMessage->exists(m:
metaRETOv01::sequenceDiagram::Message|m.oclIsTypeOf(metaRETOv01::seque
nceDiagram::Connection )implies m.target<>BEntity ))
      )
   )


then
true
else
false
endif

}


}
```

8. Bibliography

Abrahão S., Insfran, E., Genero, M. Carsí, J. A., Ramos, I. y Piattini, M. Quality-Driven Model Transformations: From Requirements to UML Class Diagrams [Journal] // Model-Driven Software Development: Integrating QUality Assurance / ed. IGI-Global. - [s.l.] : Bunse, Jorg Rech Christian, 2008. - 302-326.

Bézibvin et. al. First experiments with the ATL model transformation languaje: transforming XSLT into XQuery [Book]. - [s.l.] : 2nd OOPSLA Workshop on Generative Techniques in the context of MDA, 2003.

Bézivin J.,Gerbé,O.: Towards a precise definition of the OMG/MDA framework [Book]. - [s.l.] : .In:Proc.16thInt'lConf.AutomatedSoftwareEngineering,IEEE, 2001.

E. Insfran O. Pastor, R. Wieringa Requirements Engineering-Based Conceptual Modelling [Journal]. - [s.l.] : Springer-Verlag, 2002.

Eclipse.org Eclipse Platform Technical Overview [Online]. - Eclipse Foundation, February 2003. - 2.1. - December 2009. - http://eclipse.org/whitepapers/eclipse-overview.pdf.

Insfran E. A Requirements Engineering Approach for Object-Oriented Conceptual Modeling // Phd Thesis. - Valencia : DSIC Valencia University of Technology, 2003.

Kleppe Warmer, Bast MDA explained: the model driven architecture: practice and promise [Book]. - [s.l.] : Addison-Wesley, 2003. - Vol. 032119442X.

Koch Nora UWE, Transformation Techniques in the Model-Driven Development Process of [Book]. - [s.l.] : ICWE'06 Workshops, July 10-14, 2006, Palo Alto, CA, 2007.

Kurtev Ivan PhD Thesis. Adaptability of Model Transformations [Book]. - Enschede , the Netherlands : Febodruk BV, 2005.

Maddeh Mohamed Mohamed Romdhani, Khaled GHEDIRA MOF-EMF Alignment [Conference] // Third International Conference on Autonomic and Autonomous Systems (ICAS'07). - 2007.

Marcela Genero Esperanz aManso , Aaron Visaggio, Gerardo Canfora, Mario Piattini Building measure-based prediction models for UML class diagram maintainability [Journal]. - [s.l.] : SpringerScience BusinessMedia, LLC2007, 2007. - Vols. DOI10.1007/s10664-007-9038-4.

McNeile Ashley MDA: The Vision with the Hole? Ashley McNeile [Book]. - [s.l.] : (http://www.metamaxim.com/download/documents/MDAv1.pdf). (2003), 2003.

OMG MDA Guide V1.0 .1 [Book]. - [s.l.] : http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf, 2002.

OMG Object Management Group Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification [Online]. - april 2008. - 1.0. - 2009. - http://www.omg.org/spec/QVT/1.0/PDF.

OMG Object Management Group UML 2.0 OCL Specification [Online]. - 14 10 2003. - 2.0. - 2009. - http://www.omg.org/spec/OCL/2.0/.

Seidewitz What models mean [Book]. - [s.l.] : IEEESoftware 20, 2003.

SQuaRE ISO/IEC Software Product Quality Requirements and Evaluation (SQuaRE) [Journal]. - [s.l.] : ISO/IEC. Software Engineering, 2005.

Tom Mens Pieter Van Gorp A Taxonomy of Model Transformation [Book]. - [s.l.] : Electronic Notes in Theoretical Computer Science.

Völter Markus and Sthal Thomas Model Driven Software Development [Book]. - [s.l.] : Wiley, 2006.