

Quality-Driven Model Transformations: From Requirements to UML Class Diagrams

***Silvia Abrahão¹, Marcela Genero², Emilio Insfran¹, José Ángel Carsi¹,
Isidro Ramos¹ and Mario Piattini²***

ISSI Research Group, Department of Information Systems and Computation, Valencia University of Technology, Camino de Vera s/n, 26022, Valencia, Spain, {sabrahao, einsfran, pcarsi, iramos}@dsic.upv.es

Alarcos Research Group, Department of Information Systems and Technologies, University of Castilla-La-Mancha, Paseo de la Universidad 4, 13071, Ciudad Real, Spain, {Marcela.Genero, Mario.Piattini}@uclm.es

Abstract

Model-Driven Architecture (MDA) is a software engineering approach that promotes the use of models and model transformations as primary development artifacts. Usually, there are several ways to transform a source model into a target model. Alternative target models may have the same functionality but may differ in their quality attributes (e.g., understandability, modifiability). This chapter presents an approach to deal with quality-driven model transformations. Specifically, it focuses on a specific set of transformations to obtain UML class diagrams from a Requirements Model. A set of alternative transformations are identified, and the selection of the best alternative is done through a controlled experiment. The goal of the experiment is to empirically validate which alternative transformation produces the UML class diagram that is the easiest to understand. This evidence can be further used to define high-quality transformation processes, as it will be based on empirical knowledge rather than on common wisdom and the intuition of the researchers and developers.

1. Introduction

Nowadays, the software development community is moving towards model-driven development processes whose goal is the development of software at a higher level of abstraction based on models and model transformations. Within this context, the Model-Driven Architecture (MDA) initiative (OMG, 2003) has attracted interest from both the research community and software practitioners. 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. Here, the goal is to decouple the way, in which software systems are currently defined, which is dependant on the technology they use (OMG, 2003).

A model transformation is a process of converting one model to another model. A model may be transformed to several alternative models that may have the same functionality but different quality attributes. For example, one model may be more reusable while another model may be more comprehensive to its stakeholders. Therefore, it is necessary to identify those transformations that produce models with the desired quality attributes.

To cope with the problem of selecting alternative transformations, this chapter presents an approach for quality-driven model transformations. The mechanisms to choose the appropriate alternatives can greatly differ depending on the nature and the domain of the transformations as well as the quality perspective that is chosen. We focus on a set of transformations defined to obtain UML class diagrams from a Requirements Model (Insfran, 2003). Assuring quality in representing the system's conceptual model from requirements is particularly important, as the traceability between these models is not properly dealt with. Moreover, a conceptual model of good quality can help to minimize communication problems and misunderstandings of requirements among the stakeholders.

The quality perspective that we are interested in is the *pragmatic quality*¹ (Lindland, Sindre & Sølvsberg, 1994). This quality category addresses the

¹ There are two other types of quality according to Lindland et al.'s framework: *syntactic quality*, which is the degree to which model contains flaws, and *semantic quality*, which is the degree to which the model is valid (contains all statements in the model that are correct and relevant to the problem domain) and complete (contains all statements about the problem domain that are correct and relevant). For the purpose of our work, we focus on the evaluation of the pragmatic quality of the models obtained with alternative transformations, leaving the evaluation of the other quality perspectives as a topic for future research.

comprehension aspect of the model from the stakeholders' perspective. Pragmatic quality captures how the model has selected an alternative "from among the many ways to express a single meaning", and it essentially deals with making the model *easy to understand*.

The comprehension goal specifies that all audience members (or interpreters) completely understand the statements in the model that are relevant to them. This is an import quality attribute since it is recognized as one of the main factors that influences maintainability (Selic, 2003) (Otero & Dolado, 2004) (Reinhartz-Berger & Dori, 2005) (Genero et al., 2005; 2007). A UML class diagram must first be understood before any desired changes to it can be identified, designed, or implemented. In terms of the Lindland et al. framework, improving pragmatic quality means increasing the degree of correspondence between the set of statements in the model and the set of statements that the user thinks the model presents (i.e. their understanding of the model).

Therefore, our main goal is to empirically evaluate which of the alternative transformations produces the UML class diagram that is easiest to understand. This evidence can be further used to define high-quality transformation processes, as it will be based on empirical knowledge rather than on common wisdom and the intuition of the researchers and developers.

The structure of the chapter is as follows. Section 2 presents the state-of-the-art for quality in model-driven development. Section 3 describes how UML class diagrams can be obtained from a Requirements Model using different transformation alternatives. This section also shows the definition of these transformations using QVT and their execution in a platform for model management called MOMENT. Section 4 describes the design and the results of the experiment carried out to empirically validate the selection of the alternative transformations according to the 'understandability' quality attribute. Section 5 describes our conclusions. Finally, section 6 presents a discussion on future research directions.

2. State-of-the-Art of Quality for Model-Driven Software Development

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 1. Comparison of 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
Röttger and Zschaler, 2004	Refinement	Horizontal	Context Models	Response Time	Partial
Merilina, 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 (Merilinn 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.

3. A Quality-Driven Model Transformation Approach

This section presents a systematic approach to ensure quality in model-driven development processes. It takes a different approach to drive the selection of transformations, which is to empirically validate the selection of alternative transformations through controlled experiments. 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.

In order to operationalize this approach, we propose the use of *quality attributes* to drive the selection of the most appropriate alternative transformation that contributes to the improvement of the target model according to a given quality attribute. A quality attribute is a measurable physical or abstract property of an entity (i.e., a conceptual model) (ISO, 2001).

Currently, our controlled experiments are oriented to empirically validating the selection of the alternative transformation that maximizes the ‘understandability’ quality attribute. Fig. 1 presents an overview of our quality-driven model transformation approach.

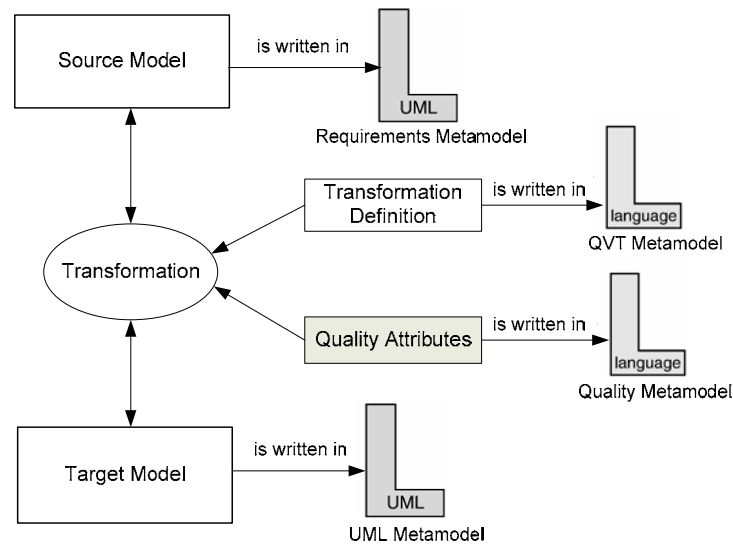


Fig. 1. Quality-driven model transformation approach

According to Fig. 1, a *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. These rules can be represented using the Query-View-Transformations (QVT) language proposed by the Object Management Group (OMG, 2005).

Another input for the transformation process is the definition of the *quality attributes* together with the corresponding empirical evidence gathered from the controlled experiments. This information will feed the transformation process with the criteria to choose the alternative transformation that maximize the selected quality attribute. Our final objective is to execute these transformations in a platform for Model Management called MOMENT (Boronat, Carsí & Ramos, 2005; 2006).

The following sections show a specific domain for applying our quality-driven model transformation approach using a Requirements Model as source model, a UML class diagram as target model, and “understandability” as the quality attribute to drive the transformations.

4. Transforming Requirements Models into UML Class Diagrams

The Requirements Model (Insfran, 2003) (Insfran, Pastor & Wieringa, 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 Tool² (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 Catalog³ (Insfran, 2003). These transformations establish traceability relationships between the Requirements Model and the UML class diagrams.

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)*”.

Fig. 2 describes a 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). For example, 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.

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>

However, the 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. If this occurs, a single alternative mapping must be properly selected according to some predefined quality attribute.

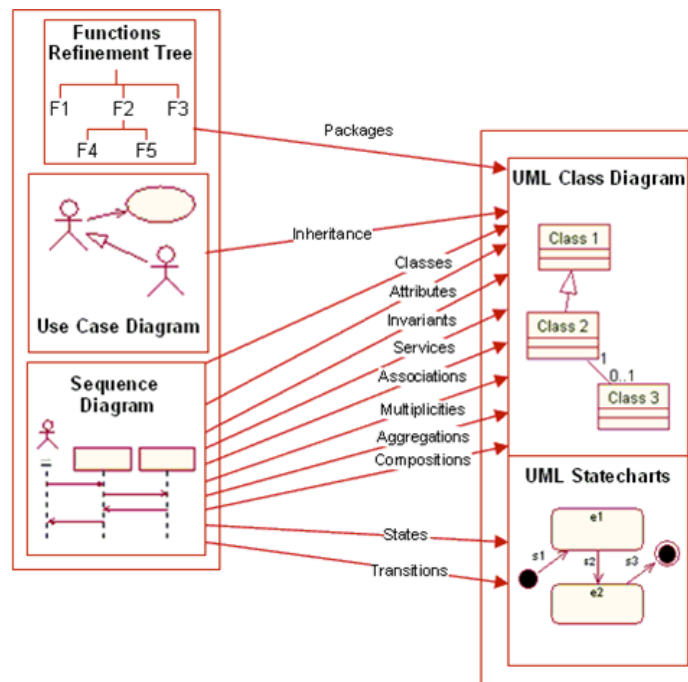


Fig. 2. Traceability from Requirements to Conceptual Models

Subsection 4.1 and 4.2 briefly introduce the requirements and the UML class diagram metamodels. The remainder of the section focus on transformations that have multiple valid representations in the UML class diagram that satisfy a given requirement pattern.

4.1 The Requirements Metamodel

Metamodeling is a key concept of the MDA paradigm and is used in Software Engineering (SE) to describe the basic abstractions that define the models and their relationships. A metamodel can be viewed as a class model whose classes and associations encode the concepts of the model and the relationships among them. The Meta Object Facility (MOF) (OMG, 2004) provides a framework for defining a metamodel and querying and manipulating the resulting models.

Fig. 3 shows 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 is specified in detail by means of one or more Sequence Diagrams. Sequence Diagrams are composed mainly of *Entities* and *Messages*. We distinguish three types of Entities when describing a Sequence Diagram: *Actor*, *Interface* and *Class*. *Actor* represents the users of the Use Case (and may or may not be a class); *Interface* represents the boundary among the actors and the internal classes of the system; *Class* represents the different entity classes that participate in the realization of the Use Case.

Finally, in order to characterize the different nature of 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.

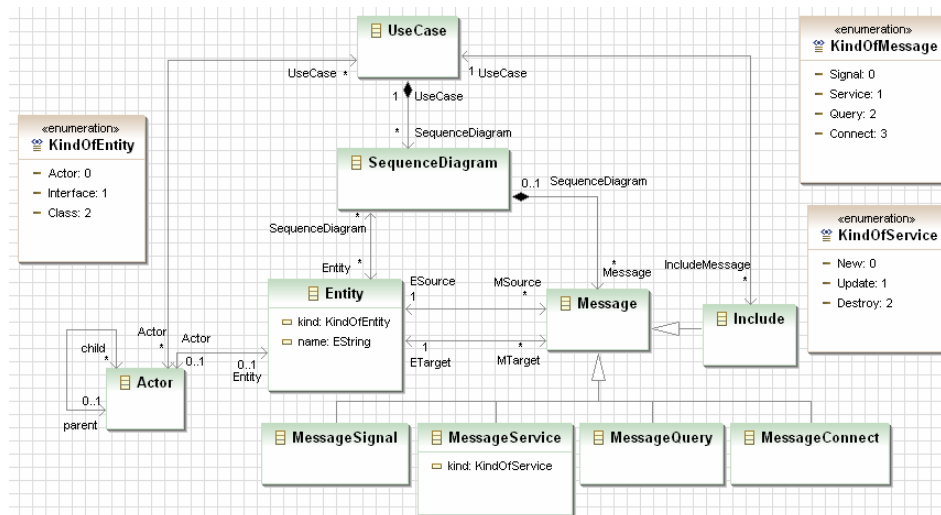


Fig. 3. Requirements Metamodel

4.2 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.

Fig. 4 shows the modified UMLite metamodel. The main part of the metamodel is the same as the metamodel defined in the QVT specification (OMG, 2005). 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.

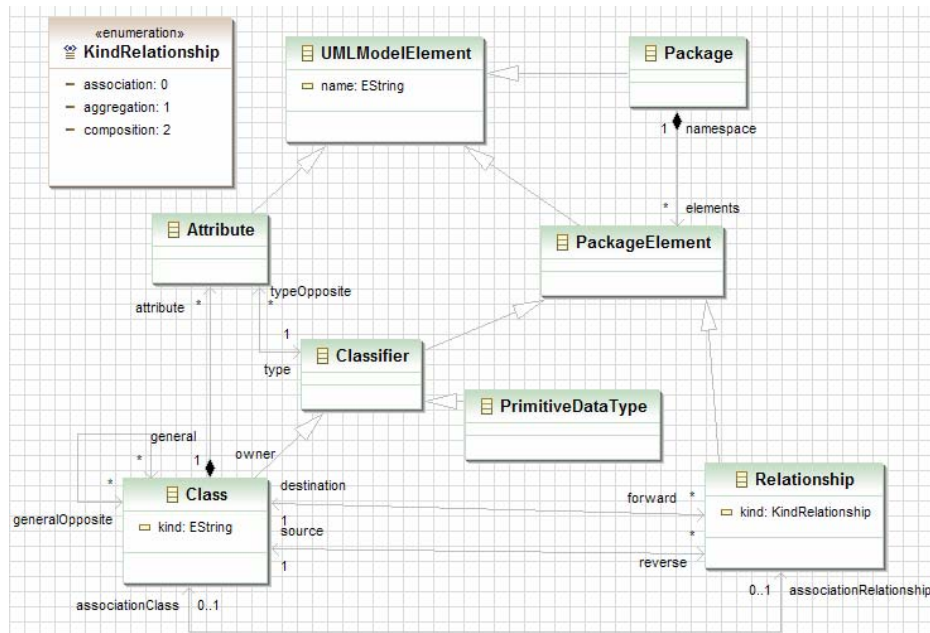


Fig. 4. The UMLite Metamodel

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.

4.3 Defining Alternative Transformations using QVT

This subsection shows how some alternative transformations for a requirement specification generate different UML class diagrams. Although not all the possibilities are fully explained due to space limitations, it is possible to see that, given a requirement specification, a set of conceptual model solutions can be identified. The example used to illustrate these alternative transformations is taken from the specification of a Car Rental system.

A Sequence Diagram is used to specify the necessary object interactions to realize the Use Case *Create Insurance* that is initiated by the *Administrator* actor. This Use Case represents the creation of a car *Insurance* policy that must be bought from an *Insurance Company* and assigned to the *Car* before using the car for rentals. Fig. 5 shows the Sequence Diagram for the Use Case *Create Insurance*. After introducing the necessary data and checking the existence of the corresponding car, a new *Insurance* object is created (messages 1 to 5). In addition, an *Insurance Company* object and a *Car* object must be connected to the new created Insurance policy (messages 6 and 7).

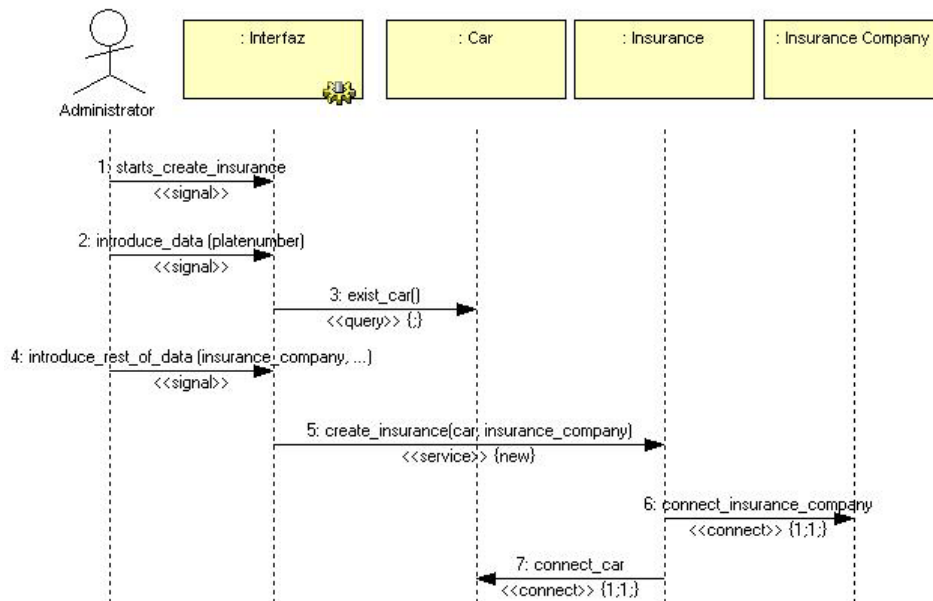


Fig. 5. Sequence Diagram showing the required interactions for the Use Case *Create Insurance*

In our approach, this information is used to transform the requirements specification into a UML class diagram⁴ following an MDA approach. It is important to remark that, for an automated transformation process to be considered useful, it must make decisions about which transformations are more suitable to produce the expected result by the analyst or a result that maximizes a quality attribute (in our case the *understandability* attribute).

The analysis of the requirement specified as object interactions shown in Fig. 5 indicates that the messages 6 and 7 satisfy the transformation rule TR 15:

- **TR15:** For every message between two classes labeled with the stereotype «connect», THEN an *association relationship* will be generated.

As a result, this transformation rule is applied twice. This means that two association relationships are established in the target model: one from *Insurance* to *Car* and another from *Insurance* to *InsuranceCompany* (Fig. 6a, **association**). An association relationship indicates a connection (link) between two classes.

⁴ The generated UML class diagram can later be modified according to traceability rules (*strong* and *weak* traceability), which are explained in (Insfran, 2003).

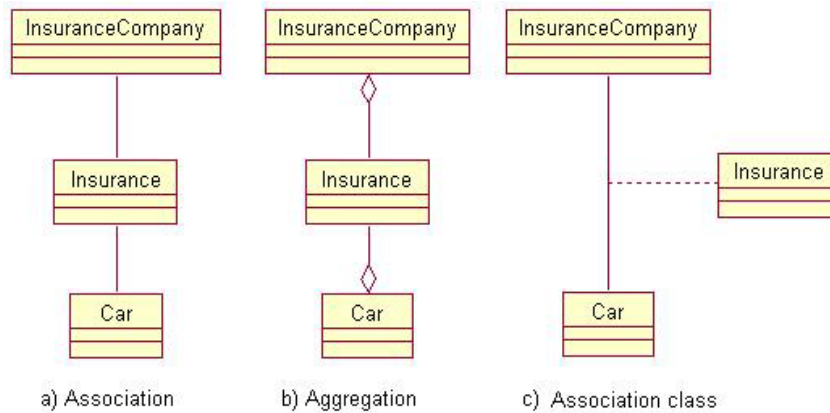


Fig. 6. Partial UML class diagram for the analyzed Sequence Diagram

Alternatively, there are other interpretations to the object interactions shown in Fig. 5. As a second alternative, the new created object *Insurance* can be represented as a component of both the *Insurance Company* compound class and the *Car* compound class (Fig. 6b, **aggregation**). This is because an aggregation is a special form of an association, specifying a whole-part relationship between two objects. This means that there is a connection between classes but also implies an additional semantics which indicates that an object ‘is made up of other objects’. We are aware that not always an association relationship can be represented as an aggregation relationship, as this decision depends on the problem domain. However, in our example, both relationships can be applied as an *Insurance* “could be *related to* an insurance company and a car” or “be *part of* an insurance company and a car”.

A third alternative is to consider the *Insurance* as an association class related to the association relationship between *Insurance Company* and *Car* (Fig. 6c, **association class**). This means that when an instance of an *InsuranceCompany* class is associated with an instance of a *Car* class, there will also be an instance of an *Insurance* class.

These three types of relationships can be alternative representations for the object interactions shown in Figure 5. In general, an association class can always be replaced by two association or aggregation relationships. Consequently, we have identified three types of structural relationships to represent object interactions (there may also be other representations). This implies the application of different transformation rules from the Transformation Rules Catalog (Insfran, 2003) to produce these relationships. Table 2 summarizes the alternative structural relationships and the transformation rules that can be applied to produce them.

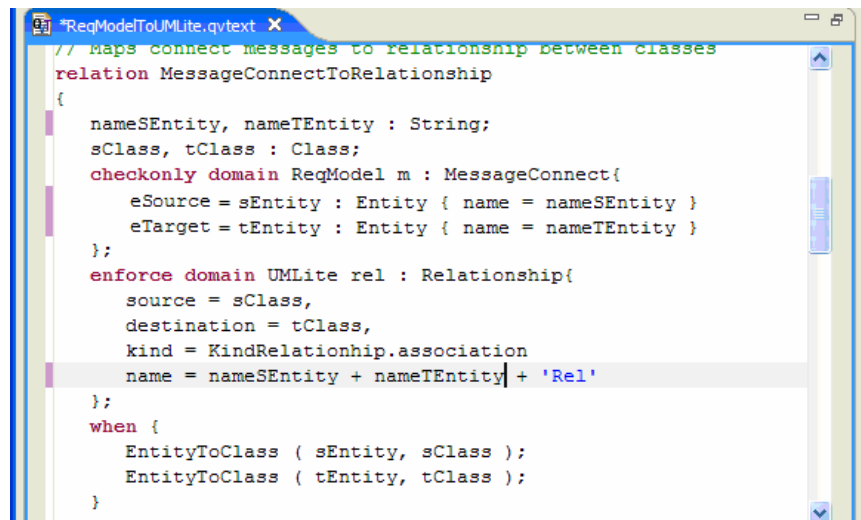
Table 2. Three alternative structural relationships and their corresponding transformation rules

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..

All the transformation rules in the catalog are being specified using the declarative QVT relations language (OMG, 2005). Fig. 7 shows the specification of the transformation rule TR15 in QVT.

4.4 Executing the Alternative Transformations in MOMENT

MOMENT (Boronat, Carsí & Ramos, 2005) is a framework for model management that is fully integrated in the Eclipse environment. MOMENT combines the best features of Maude and Eclipse. First, it uses Maude (Clavel et al., 2005) as a backend. Maude is a reflective language and system supporting equational and rewriting logic specification. Maude has been used as a rapid prototyping environment to develop MOMENT using some of its properties: pattern matching, parameterization, and reflection. Second, Eclipse Modeling Language (EMF) is an industrial standard that includes a metamodel (Ecore) to define, modify, and serialize models with a very efficient reflexive API.



```
*ReqModelToUMLite.qvtext X
// Maps connect messages to relationship between classes
relation MessageConnectToRelationship
{
    nameSEntity, nameTEntity : String;
    sClass, tClass : Class;
    checkonly domain ReqModel m : MessageConnect{
        eSource = sEntity : Entity { name = nameSEntity }
        eTarget = tEntity : Entity { name = nameTEntity }
    };
    enforce domain UMLite rel : Relationship{
        source = sClass,
        destination = tClass,
        kind = KindRelationship.association
        name = nameSEntity + nameTEntity + 'Rel'
    };
    when {
        EntityToClass ( sEntity, sClass );
        EntityToClass ( tEntity, tClass );
    }
}
```

Fig. 7. TR15 specified using QVT-Relations

To use MOMENT to apply transformations (Boronat, Carsí & Ramos, 2006), the source metamodel (the Requirements Metamodel) and the target metamodel (the UMLite class diagram Metamodel) must be defined and registered as Ecore models. MOMENT is used to define the QVT-Relations transformation *ReqModelToUMLite*. This transformation is composed of a set of rules that defines how to transform information belonging to a model that conforms to the Requirements Metamodel into a model that conforms to the UMLite class diagram Metamodel.

Finally, a configuration for the MODELGEN operator is defined. This configuration is composed of the *ReqModelToUMLite* transformation, a source model that is defined with the RETO tool, and the name of the target UML model. Additionally, a traceability model is generated to relate the elements of the source model with the elements of the target model.

Fig. 8 shows MOMENT integrated in the Eclipse environment. On the left side, the Package Explorer shows all the files related to the transformation: *ModelGen.mop* is the model that defines the generic operator MODELGEN; *reqModel.ecore* is the model that defines the Requirements Metamodel used as the source in the transformation; *UMLite.ecore* is the metamodel used as the target in the transformation; *rentacar.reqmodel* is a model that defines the rentacar system (instance of the metamodel reqModel); *rentacar.umlite* is the model (instance of UMLite metamodel) resulting from the application of the transformation to *rentacar.reqmodel*; *rentacar.traceabilitymodel* is the traceability model that maps elements of the source and target models; *ReqModelToUMLite.qvtext* is the transformation expressed in QVT-Relations. The top of the figure shows part of the UMLite metamodel. The bottom of the

figure shows a part of the ReqModelToUMLite transformation inside the QVT text editor.

An additional transformation called *UMLite2UML2* was defined to transform UMLite class diagram models into UML2 class diagram models (see section 3.2).

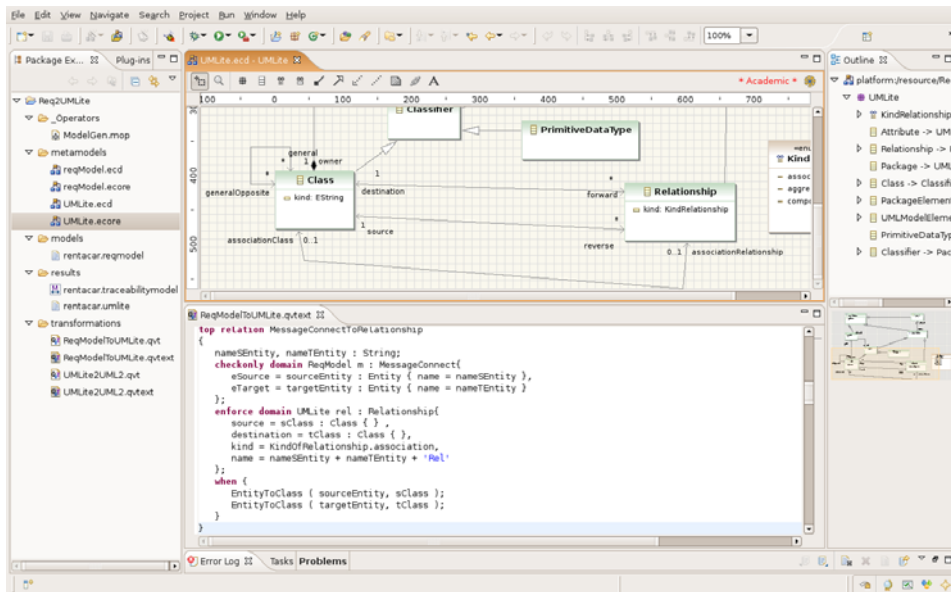


Fig. 8. MOMENT environment

Once a transformation has been executed, a traceability model is generated. The traceability model relates the elements of both models. There is a special view designed for this information. This view allows the analyst to see the transformation rules that have been executed and what the results are.

Fig. 9 shows the traceability model generated after executing the *reqModel2UMLite* transformation. The first column shows the domain model, which is the *rentacar* requirements model. The third column shows the range model, which is the generated *rentacar* UMLite model. Finally, the second column shows the traceability links (mappings) that relates the elements of the domain and range models. A traceability link, which is the result of applying the transformation rule *EntityToClass* is highlighted. This traceability link relates the *Insurance* entity with the *Insurance* class.

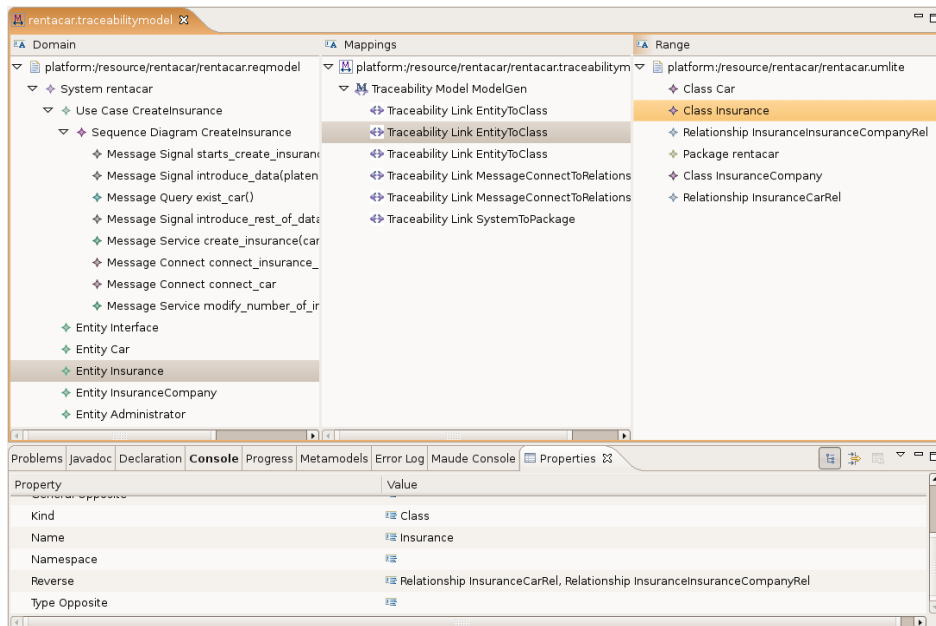


Fig. 9. Traceability model generated by applying the *ReqModelToUMLite* transformation.

5. Experiment Description

This section presents a description of the experimental process that was followed to select the best alternative transformation. The process is based on the experimental frameworks proposed by (Wohlin et al., 2000; Juristo & Moreno, 2001). This process is composed of the following activities: definition, planning, operation, and analysis and interpretation.

5.1 Definition

The main goal of this experiment is to determine which of the transformation rules for structural relationships between classes introduced in section 4.3 (A1, A2, A3, and A4) obtained the easiest to understand UML class diagram. Therefore, using the GQM (Basili & Rombach, 1988) template for goal definition, the goal of our experiment is defined as follows:

Analyze	<i>Alternative transformation rules for structural relationships between classes (A1, A2, and A3)</i>
For the purpose of	<i>Evaluating</i>
With respect to	<i>the Understandability of the obtained UML class diagrams</i>
From the point of view of	<i>the researchers</i>

In the context of *Undergraduate students at the Department of Information Systems and Computation at the Valencia University of Technology*

5.2 Planning

The next step is planning. The definition determines *why* the experiment is conducted, while the planning prepares *how* the experiment is to be conducted. The main characteristics of the planning phase are the following:

Subjects. The participants were 39 fourth-year students in Computer Science at the Valencia University of Technology, who were taking part in the second Software Engineering course. We took a “convenience sample” (i.e. all the students in the class). The subjects had six months of experience in modeling with UML and three years of experience in the OO paradigm. The subjects were encouraged to participate by offering them an extra point in the final grade for performing the required tasks correctly.

Variable selection. The independent variables were the transformation rules for structural relationships between classes (i.e., A1, A2, and A3). The dependent variable was understandability.

Experimental material and tasks. The experimental material and tasks consisted of:

- 9 Sequence Diagrams from three different case studies, with 3 UML class diagrams each. These were obtained by applying the alternative transformation rules. An example of the experimental material is shown in Appendix A. The rest of experimental material is available at: www.dsic.upv.es/~einsfran/experiment.
- Each Sequence Diagram has a questionnaire attached consisting of 6 Yes/No questions to test the subjects’ understanding of the Sequence Diagrams. The effectiveness of the subjects in answering the questionnaires (number of correct answers/number answers) was used as a criterion to exclude those observations that did not fulfill a minimum level of quality. Observations with a value less than or equal to 0.5 were excluded. If the subjects did not understand the Sequence Diagrams, their questionnaires were excluded.
- Each of the three UML class diagrams had a questionnaire attached (with 6 questions) for assessing which alternative UML class diagram was better understood by the subjects. In addition, the subjects had to write down the starting and ending times for completing the questionnaires. We obtained three measures for understandability from this understanding task:
 - *Understandability Time*, which reflects the time, in seconds, that the subjects spent answering each questionnaire (calculated by the difference between the ending time and the starting time). Each subject

completed 3 questionnaires detailing 3 alternatives (A1, A2, and A3). Three understandability time measures (A1Time, A2Time, and A3Time) were obtained.

- *Effectiveness*, which reflects the correctness of the answers (calculated by dividing the number of correct answers by the number of answers). Three understandability effectiveness measures (A2Effec, A2Effec, and A3Effec) were obtained.
- *Efficiency*, which reflects the correctness of the answers by time (calculated by dividing the number of correct answers by the understandability time). Three measures for understandability efficiency (A2Effic, A2Effic, and A3Effic) were obtained.
- The final task of each test consisted of asking the subjects which of the three alternative UML class diagrams best reflected the problem modeled in the Sequence Diagram. In this way, we obtained a subjective measure (*Alternative Selected*) based on the subjects' perception.

Hypothesis formulation. The following hypotheses were formulated:

- **H1₀**: The use of different alternative transformations (A1, A2, and A3) does not affect the Understandability Time (A1Time, A2Time, and A3Time). $H1_1 = \neg H1_0$
- **H2₀**: The use of different alternative transformations (A1, A2, and A3) does not affect the Understandability Effectiveness (A1Effec, A2Effec, and A3Effec). $H2_1 = \neg H2_0$
- **H3₀**: The use of different alternative transformations (A1, A2, and A3) does not affect the Understandability Efficiency (A1Effic, A2Effic, and A3Effic). $H3_1 = \neg H3_0$
- **H4₀**: There is no correlation between the Alternative Selected and the means of objective Understandability variables (Understandability Time Effectiveness, and Efficiency). $H4_1 = \neg H4_0$

5.3 Operation

The experiment started with an introductory session in which the main concepts of the Requirements Model (e.g., the notation of Sequence Diagrams) were reviewed. The goal of the experiment was not disclosed to the subjects. Then, the subjects were shown an example of the experimental material, which was similar to what they would be using during the execution of the experiment.

Each subject was given all the experimental material, including nine tests (balanced within-subject design). The diagrams were assigned in different order to limit learning effects. The alternatives were also organized in a different order across subjects in order not to favor one alternative over another. In total, eighteen types of tests were prepared. The subjects were instructed how to

develop the experimental tasks and they had a maximum of two hours to complete all the tasks.

5.4 Data Analysis and Interpretation

After the experiment took place, we collected the experiment data. It consisted of a table of 351 rows (9 diagrams x 39 subjects) and 9 columns (A1Time, A2Time, A3Time, A1Effec, A2Effec, A3Effec, A1Effic, A2Effic, A3Effic). We then performed a “data cleaning”, excluding the observations that were not complete because the subjects had not written down the time or because the subjects did not selected the best alternative. Since all the questions in each questionnaire were complete, the completeness of the performed tasks was guaranteed. We also excluded the observations that had a value of effectiveness of 50% or less for each Sequence Diagram. The final data for testing the hypotheses were 325 observations.

The following statistical analyses were performed to analyze the data:

- A descriptive study was done to characterize the dependent variables.
- Hypotheses H1, H2, and H3 were tested using an ANOVA test with repeated measures.
- Hypothesis H4 was tested using the Spearman correlation coefficient.

We used SPSS (SPSS, 2002) to carry out the data analyses presented in this study.

5.4.1 Descriptive statistics

The descriptive study was performed by first analyzing the variable *Alternative Selected* and then analyzing the measures of *Understandability Time*, *Effectiveness*, and *Efficiency*.

From Table 3 (which shows the frequency of each type of alternative (A1, A2, and A3) for each diagram) and Fig. 10 (which shows the percentages of selection for each type of alternative) we can infer the following:

- A1 is the alternative transformation that was most selected by the subjects, i.e., the subjects believed that the use of associations allowed to obtain the best UML class diagram (the easiest to understand).
- A3 is the alternative transformation that was least selected by the subjects, i.e., the subjects believed that it was the least appropriate alternative transformation.

Table 3. Frequency of transformation alternatives per diagram

Diagrams/ Alternatives	D1	D2	D3	D4	D5	D6	D7	D8	D9	Total
A1	6	12	13	25	18	14	16	18	7	129
A2	8	4	7	8	9	12	6	5	13	72
A3	19	20	10	5	11	12	15	14	18	124
Total	33	36	30	38	38	38	37	37	38	325

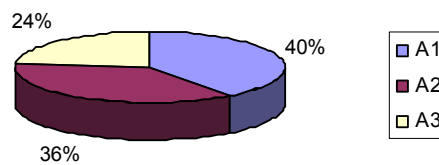


Fig. 10. Percentages for Alternative Selected

The descriptive statistics for the *Understandability Time*, *Effectiveness*, and *Efficiency* are shown in Table 4. They are ranked in ascendant order by the value of the mean.

Table 4. Descriptive statistics for the measures for Understandability Time, Effectiveness and Efficiency.

	Min.	Max.	Mean	St. Dev.
A2TIME	23	449	93.4338	57.9701
A1TIME	12	611	101.9046	79.1776
A3TIME	15	734	106.5077	72.2032
A2EFFEC	0.167	1	0.8701	0.1528
A3EFFEC	0.167	1	0.8844	0.1821
A1EFFEC	0.167	1	0.9111	0.1548
A3EFFIC	0.006	0.4	0.0677	0.0419
A2EFFIC	0.011	0.24	0.0757	0.0436
A1EFFIC	0.005	0.5	0.0803	0.0531

Table 4 reveals that, on average, the subjects spent less time performing the tasks related to alternative A2. However, the difference with the other tasks was not very significant (approximately 8 seconds for A1 and A3). The subjects

were more effective and efficient performing the tasks related to alternative A1; but the difference in effectiveness with the other alternatives was not very significant.

In summary, the descriptive statistics show a slight tendency in favor of A1, which is the transformation based on associations.

5.4.2 Testing Hypotheses

To test the hypotheses H_1 , H_2 , and H_3 , we carried out an ANOVA for repeated measures. The results show hypotheses H_{1_0} , H_{2_0} , and H_{3_0} can be rejected (with a significance level = 0.05). This means that each alternative transformation really does affect the *Understandability Time*, *Effectiveness*, and *Efficiency*.

Moreover, we compared the means for each measure by pairs of alternatives. There was a significant difference between the following pairs:

- The pairs A1-A2, A1-A3 in the values of the *Understandability Time*.
- The pairs A1-A2, A1-A3 and A2-A3, in the *Understandability Effectiveness*.
- The pairs A1-A3, A1-A3 and A2-A3, in terms of the *Understandability Efficiency*.

This comparison shows that there is a significant difference between A1 (related to associations) and the other alternatives (A2 and A3).

To test hypothesis H_4 , we carried out a correlation analysis using the Spearman Correlation, separately per each diagram. We did not find any correlation between the subjective measure (*Alternative Selected*) and the mean of the objective measures (*Understandability Time*, *Effectiveness*, and *Efficiency*). This reveals that the use of an alternative transformation is not dependent on how effective or efficient the subjects are (i.e., the performance of the subjects did not affect their perception).

In summary, the main findings of the experimentation show that there exists a slight tendency in favor of using associations. In other words, the subjects are a slightly more effective and efficient when performing tasks related to association relationships (instead of aggregations or association classes). Assuming that the three alternatives are alternatives, this indicates that transformations related to association relationships are the most appropriate when the understandability quality attribute is selected.

5.5 Threats to validity

This section discusses several issues that can affect the validity of the empirical study and how we attempted to alleviate them.

In order to control the risk that the variation due to individual differences is larger than due to the treatment, we selected a homogeneous group of subjects. In addition, to attempt to control the internal validity of the study, the following issues were considered:

- *Differences among subjects.* Using a within-subjects design, error variance due to differences among subjects was reduced. In addition, we randomly assigned the tests to the subjects in different order. This procedure cancels out a possible learning effect (due to similarities in the treatments) and a confounding effect (due to the order in which the alternatives were presented).
- *Knowledge of the universe of discourse.* We used the same requirement specification document for all subjects. It specifies the requirements of a Car Rental System for a company. This is a well-known universe of discourse.
- *Fatigue effects.* On average, each subject took two hours to solve the experimental tests, so fatigue was not very relevant.
- *Persistence effects.* In order to avoid persistence effects, the experiment was carried out by subjects who had never done a similar experiment.
- *Subject motivation.* We motivated students to participate in the experiment by offering them an extra point in the final grade of the course.

One limitation to the external validity of this study is the fact that the three alternative transformation rules cannot be applied simultaneously to all modeling situations. For instance, to establish an association class relationship (A4), at least one «service/new» message and two «connect» messages are needed in the source model. The goal of this experimentation was to gather empirical evidence for the specific case when the three alternative transformations can be applied to obtain a relationship between classes. We are aware that, more alternatives may be possible to represent structural relationships between classes. More experimentation is needed to validate these other combinations.

Another limitation is the use of only students as participants. In general, our students have no working experience in conceptual modeling. The use of student participants may present a threat to the study's external validity. However, the students who participated in the experiment were fourth-year students in Computer Science. Therefore, they can be considered as representative of novice users of conceptual modeling approaches. To increase external validity, the current study needs to be replicated using experienced practitioners from the industrial sector who are experienced in UML and/or students with higher levels of training in order to confirm our results.

6. Conclusion

This chapter has presented an approach for quality-driven model transformations. Specifically, it described a controlled experiment to investigate the selection of alternative QVT transformations to obtain UML class diagrams from a Requirements Model. The goal of the experiment was to gather empirical evidence about which alternative transformation produces the UML class diagram that is easiest to understand.

The results show that there is a slight tendency to favor the use of association relationships when the three alternatives can be applied. This indicates that transformations related to association relationships are the most appropriate when the understandability quality attribute is selected. A possible reason for this could be that this relationship has less semantic strength than the other kinds of relationships. When an aggregation relationship is chosen instead of an association relationship, analysts know that they are defining a *part-of* relationship. However, when an association class is chosen, the same relationship can be represented using two association or aggregation relationships.

These results provided first evidence about the understandability of a model obtained through a model transformation process. The study was conducted in the context of the UML class diagram that is the most-used specification for model-driven software development in industry. Although this evidence is specific to this domain, and in particular, to the relationships among classes, it should be generalized to other elements in a UML class diagram, other UML diagrams, and also to other domains. Therefore, more experimentation is needed to verify the generalizability of our approach.

The results that we have obtained through experimentation are promising. However, they must be considered as preliminary results. We plan to replicate this experiment with students from the University of Castilla-La Mancha in Spain and also with more experienced practitioners from the industrial sector in order to confirm these results. We believe that the level of experience in UML modeling can considerably influence the performance of the subjects.

7. Future Research Directions

Our literature review has shown that there are very few studies that deal with quality in model-driven development. As far as we know, there is no study encompassing the empirical validation of model transformations. The study of quality for model-driven development is of great relevance to software development organizations faced to the adoption of this technology in industry.

The empirical study presented in this chapter shows how model transformations can be empirically validated with regard to a given quality attribute. This work is part of a project on quality-driven model transformations whose goal is the

definition of a quality metamodel to drive the selection of model transformations according to multiple quality attributes. Therefore, our current research efforts are focused on the validation of the remaining transformations of the Transformation Rules Catalog (Insfran 2003). We also plan to study other quality attributes (i.e., efficiency, understandability, usability, modifiability) and possible conflicts that could arise when more than one quality attribute is chosen. Our ultimate goal is to build an empirically validated quality metamodel to drive the selection of model transformations in different domains (e.g., Bioinformatics and Data Warehouse). This quality metamodel will be fully integrated in the MOMENT environment.

There is an urgent need for more research studies of this type to complement and extend the current empirical study. Empirical evaluation of model transformations will help software developers assess the usefulness of different sets of transformations according to the quality of the resulting target model and/or transformation needs.

While several studies (including this one) have studied model transformations and its properties, it would be interesting to survey or interview domain-specific engineers and ascertain the importance of certain model transformations to define heuristics to drive the transformations. In our study, a heuristic could be the type of traceability (i.e., strong and weak) assigned to each transformation rule. This could be an additional criterion to be used during the transformation process.

Another area of future research needs to examine other quality perspectives (i.e., syntactic quality, semantic quality). Syntactic quality in the context of model-driven development is trivial due to all the models are compliant to their respective metamodel. However, assessing semantic quality will allow to verify which alternative transformation will produce a target model that is more correct and relevant to the problem domain.

Finally, the applicability or dependence of model transformations to the type of domain of the application being developed would be an interesting study.

References

- Basili, V. & Rombach, H. (1988). The TAME project: towards improvement-oriented software environments. *IEEE Transactions on Software Engineering*, 14(6), 728-738.
- Boronat, A., Carsí J.A., & Ramos I. (2006). *Algebraic Specification of a Model Transformation Engine*. Proceedings of the Fundamental Approaches to Software Engineering (FASE'06). ETAPS'06. Vienna, Austria, 262–277.
- Boronat, A., Carsí, J.Á., Ramos, I. (2005). *MOMENT: a formal MOdel manageMENT tool*. School on Generative and Transformational Techniques in SE. Braga, Portugal.
- Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., & Talcott, C. (2005). Maude 2.2 manual and examples, from <http://maude.cs.uiuc.edu/maude2-manual>

Silvia Abrahão, Marcela Genero, Emilio Insfran, José A. Carsí, Isidro Ramos & Mario Piattini, *Quality-Driven Model Transformations: From Requirements to UML Class Diagrams*

Czarnecki, K., & Helsen, S. (2006). Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3), 621–645.

Genero, M., Manso, M., Visaggio, A., Canfora, G. & Piattini, M. (2007) Building measure-based prediction models for UML class diagram maintainability. *Empirical Software Engineering*, (to appear).

Genero, M., Moody, D., & Piattini, M. (2005). Assessing the capability of internal metrics as early indicators of maintenance effort through experimentation. *Journal of Software Maintenance and Evolution: Research and Practice*, 17, 225-246.

Insfran, E. (2003). *A Requirements Engineering Approach for Object-Oriented Conceptual Modeling*, PhD Thesis, DSIC, Valencia University of Technology, Spain.

Insfran, E., Pastor, O. & Wieringa, R. (2002). Requirements Engineering-Based Conceptual Modelling. *Journal of Requirements Engineering*, 7 (2), 61–72, Springer-Verlag.

ISO, ISO/IEC 9126-1, (2001). Software Engineering – Product quality – Part 1: Quality model.

Ivkovic, I., & Kontogiannis, K. A. (2006). *Framework for Software Architecture Refactoring using Model Transformations and Semantic Annotations*, Proc. of the Conference on Software Maintenance and Reengineering (CSMR'06), 135–144.

Jurista, N. & Moreno, A. M. (2001). *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers.

Kerhervé, B., Nguyen, K. K., Gerbé, O., & Jaumard, B. A. (2006). Framework for Quality-Driven Delivery in Distributed Multimedia Systems, Proc. of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006), 195–205.

Kurtev, I. (2005). *Adaptability of Model Transformations*. PhD Thesis, University of Twente, The Netherlands.

Lindland, O. I., Sindre G., & Sølvberg A. (1994). Understanding quality in conceptual modeling. *IEEE Software*, 11(2), 42–49.

Markovic, S., & Baar, T. (2005). *Refactoring OCL annotated UML class diagrams*. In Proc. of the 8th Int. Conference on Model Driven Engineering Languages and Systems, 280–294.

Merilinna, J. (2005). *A Tool for Quality-Driven Architecture Model Transformation*. Espoo, VTT Electronics, VTT Publications.

OMG, (2006). OMG, UML 2.1 Unified Modeling Language™

OMG, (2005). OMG, MOF 2.0 Query/Views/Transformations Final Adopted Specification, Object Management Group, from <http://www.omg.org/cgi-bin/apps/doc?ad/05-11-01.pdf>

OMG, (2004). Meta Object Facility (MOF) 2.0 Core Specification, ptc/04-10-15.

OMG, (2003). MDA Guide, from <http://www.omg.org/docs/omg/03-06-01.pdf>. Version 1.0.1.

Otero, M. C., & Dolado, J. J. (2004). Evaluation of the Comprehension of the Dynamic Modeling in UML. *Information and Software Technology*, 46(1), 35-53.

Silvia Abrahão, Marcela Genero, Emilio Insfran, José A. Carsí, Isidro Ramos & Mario Piattini, *Quality-Driven Model Transformations: From Requirements to UML Class Diagrams*

Reinhartz-Berger, H. & Dori, D. (2005). OPM vs. UML—Experimenting with Comprehension and Construction of Web Application Models. *Empirical Software Engineering*, 10, 57–79.

Rottger S., & Zschaler, S. (2004). Model-Driven Development for Non-functional Properties: Refinement through Model Transformation, In LNCS Volume 3273, The Unified Modelling Language (UML) Conference, pp. 275–289.

Selic, B. (2003). The Pragmatics of Model-Driven Development. *IEEE Software*, 20 (5), 19-25.

SPSS, SPSS 11.5, Syntax Reference Guide. 2002, SPSS Inc.: Chicago, USA.

Sottet, J. S., Calvary, G., & Favre, J. M. (2006). Mapping Model: A First Step to Ensure Usability for sustaining User Interface Plasticity, In: Proc. of the MODELS 2006 Workshop on Model Driven Development of Advanced User Interfaces.

Wohlin C., Runeson P., Höst M., Ohlson M., Regnell B. and Wesslén A. (2000). *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers.

Zou, Y., Kontogiannis, K. (2003). Quality Driven Transformation Framework for OO Migration. In. *Proc. 2nd ASERC Workshop on Software Architecture*, Banff, Canada, pp. 18–24.

Additional Reading

Endres, A. & Rombach, D. (2003). *A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories*. Addison Wesley.

Frankel, D. S. (2003) *Model driven Architecture: Applying MDA to Enterprise Computing*, Wiley.

Juristo, N. & Moreno, A. M. (2001). *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers.

Maxwell, K. (2002). *Applied Statistics for Software Managers*. Software Quality Institute Series. Prentice Hall.

Stahl, T., Voelter, M., & Czarnecki, K. (2006) *Model-Driven Software Development: Technology, Engineering, Management*, Wiley.

Unhelkar, B. (2005) *Verification and Validation for Quality of UML 2.0 Models*, Wiley-Interscience.

Wohlin C., Runeson P., Höst M., Ohlson M., Regnell B. and Wesslén A. (2000). *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers.

Appendix A. An Example of the Experimental Material

TEST R1

The following Sequence Diagram represents the creation of a *Car* for a car rental company. All the cars of the company have an assigned *Rate*. In addition, they must have an *Insurance* policy from an *Insurance Company*.

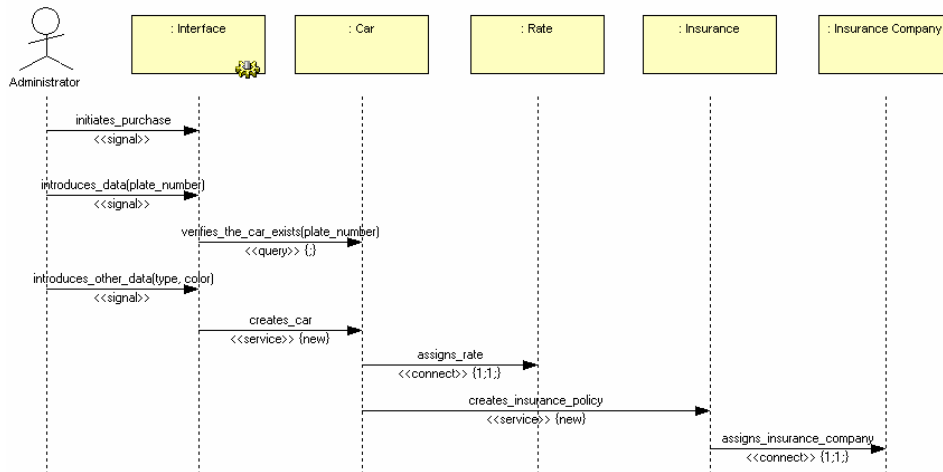


Fig. 11. Sequence diagram “creation of a car”

SECTION A: Understandability

WRITE DOWN THE CURRENT TIME (HH: MM: SS) _____

Answer the following Yes/No questions:

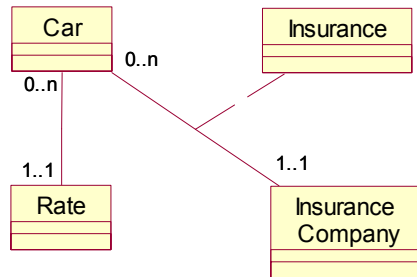
1. Is it possible in this scenario to create several *Cars*? _____
2. Can a *Car* have several *Insurance* policies in addition to the obligatory one for accidents?; for example, a policy for theft _____
3. Are there four classes in this Sequence Diagram? _____
4. If an appropriate *Rate* for a *Car* does not exist, can a new type of *Rate* be created and then assigned to the *Car*? _____
5. Can a *Car* be created without an obligatory *Insurance* policy? _____
6. Is it possible to associate an *Insurance* policy of another *Car* to the *Car* being created? _____

WRITE DOWN THE CURRENT TIME (HH: MM: SS) _____

Section B: Alternatives of Representation

Note: As an example of the three alternatives obtained from the Sequence Diagram shown in Fig. 11, we include the one based on the association class relationship.

Alternative 1:



WRITE DOWN THE CURRENT TIME (HH: MM: SS) _____

Answer the following Yes/No questions:

1. Does the *Insurance* policy exist because there is a relationship between the *Car* and *Insurance Company* classes? _____
2. Can an *Insurance* policy be related to an *Insurance Company* without being related to the *Car*? _____
3. If the relationship between the *Car* and the *Insurance Company* is destroyed, can the *Insurance* policy continue to exist? _____
4. If the *Insurance* policy is destroyed, must the relationship between the *Car* and the *Insurance Company* also be destroyed? _____
5. If the *Insurance* policy is destroyed, must the *Rate* be destroyed too? _____
6. _____ Can a *Car* have several *Insurance* policies? _____

WRITE DOWN THE CURRENT TIME (HH: MM: SS) _____

...

Note: For reasons of brevity, only Alternative 1 is shown. See www.dsic.upv.es/~einsfran/experiment for the alternatives 2-3 (in Spanish).

SECTION C: Rating tasks

In your opinion, which one of the UML class diagrams presented in Section B best represents the scenario illustrated in the Sequence Diagram of Fig. 11? (Mark your choice with an "X")

Alternative 1 ()

Alternative 2 ()

Alternative 3 ()