TECHNISCHE UNIVERSITÄT BERLIN
FAKULTÄT IV ELEKTROTECHNIK UND INFORMATIK
BACHELORSTUDIENGANG INFORMATIK

WILLIAM BOMBARDELLI DA SILVA

Towards Synchronizing Relations Between Artifacts in the Java Technological Space

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr.

Berlin
December 2015

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den

....
Unterschrift

ACKKNOWLEDGMENTS

....

ABSTRACT

...

Keywords: 1. 2.

.

ZUSAMMENFASSUNG

.

Schlagwörter: ..

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

TGG	Triple graph grammar

CONTENTS

1 INTRODUCTION

The techniques for software development has changed in the course of time since the rise of general-purpose programmable computers and specially in the second half of the 20th century with the rise of digital computers (Ceruzzi, 2003). In the beginning of digital computer programming machine code was used to describe algorithms, but as complexity and size of such algorithms got bigger this technique soon became impracticable, what evoked the need for a more sophisticated way of programming these digital machines. The assembly languages (also known as low-level programming languages) came to solve this problem, but clearly the complexity kept increasing as well as the need for new techniques and technologies for software programming. The popularization of computing, and the increasing application of computers in the practice urged the creation of high-level programming languages (e.g. Cobol, Fortran), which kept evolving mainly in regard to the needs of the software market (Ceruzzi, 2003). More sophisticated languages (e.g. C, Pascal) and new paradigms (e.g. modular and object-oriented programming) also arose in the late 20th century. But the evolution of software development does not seem to stop, evidenced by the lately increasing research on new software engineering techniques such as the Model-driven Engineering.

The newest characteristics of the information system market, like the constant evolution of the software systems, the interoperability between them and the big number of developers working in a common software artifact has required the use of software models; and the research on how to apply systematically and correctly such models in the development processes, what is called Model-driven Engineering or Model-driven Software Development France and Rumpe (2007). This Bachelor thesis aims therefore to explore one specific realm of Model-driven Engineering research, namely the problem of synchronization of models (or artifacts) in the Java technological space. The goal here is to pick commonly used meta-models in Java systems, describe them and identify their relations, so that they can be synchronized.

TALK ABOUT THE REMAINDER OF THE DOCUMENT.

1.1 Background

According to Czarnecki and Helsen (2006, p. 21): "*Models are system abstractions that allow developers and other stakeholders to effectively address concerns, such as answering a question about the system or effecting a change*". By defining a model as a system abstraction, it becomes clear, that a software system might have several models abstracted from it, each one representing certain aspects of the whole system. These models also have relations between them, in the sense that they all are supposed to describe the actual system consistently by not presenting logical contradictions. Here examples of models are *UML class diagram*, *Use Cases*, or even the source-code itself. The term model and artifacts will be used interchangeably throughout this document.

The constant evolving nature of current large-scale software systems causes their models to be constantly changed Diskin (2011). But in order to maintain this whole network of interconnected models consistent the changes have to be forwarded through the network, i.e. all models have to be synchronized. Suppose one have a *UML Class Diagram*, a series of *UML Sequence Diagrams* and the source-code. If a method has its name changed in the class diagram, all occurrences of this method has to have its name updated in the sequence diagrams and in the source-code. It turns out though, that neither a model synchronization tool comprising the most common meta-models used nowadays in Java information systems is known by us nor have we found clearly defined relation definitions between them on the literature.

1.2 Objective

The goal of this bachelor thesis is first to choose software meta-models commonly used in current Java object oriented software systems, writing down the meta-model definitions; second to define the relations between the chosen meta-models; and last, to showcase how these artifacts can be synchronized in some representative cases. We work therefore with the hypothesis, that such meta-models can be found and defined; that some relations between them can be written in some language; and that some of these relation can be synchronized using a tool or technique available in the current literature.

The end of this document shall present all the written definitions as well as the results of the synchronization of representative examples useful in the practice. Furthermore, the report of this thesis recording the difficulties and experiences found during the work process and an analysis and discussion of future development and challenges of the realm is also a legacy.

1.3 Motivation

The lack of a functional model synchronization tool integrating a broad range of meta-models used currently in Java software engineering is the main motivation for this thesis. Although an expressive effort has been made by the academic community to create solutions for the problem of model synchronization, no study known by us presents an effective tool, that could be used extensively in practice. We believe though, that the contribution of this thesis can be useful for the creation of such a tool.

Another motivation for this thesis is the lack of relation definitions in current literature for extensively used meta-models in Java systems industry. Examples of these relations include relations between *UML Class and Sequence Diagrams* and *Java Code*, between *Use Cases* and *Requirements Diagrams*, between *OCL contracts (used in design by contract methods)* and *Unit Tests*, among others. It means, that the success of this thesis brings the contribution towards the definition of a network of interconnected meta-models useful to both research and industry community. Therefore the availability of such a network might finally allow the extensive use of Model-driven Engineering in practice – helping bridging the gap between system abstractions and their concrete form – and foster the further development of more sophisticated model synchronization methods.

It is worth to note also, that the contribution of this thesis might help enhancing the quality of current software construction and therefore lessening the number of software problems and errors, what seems to be a big problem nowadays, by fomenting the wide application of Model-Driven Software Development.

1.4 Methodology

In order to achieve the goals the following procedure is taken. In the first moment a collection of common meta-models used in the Java technological space is to be defined, this is done through an state-of-art research, since that some meta-models have already been defined by other authors, plus the creation of our own versions of some meta-models, that are not available in the state-of-art. So for example, in this phase the choice of the applied meta-models (i.e. *UML Class Diagram* and *Java Code*) will be done and their definitions will be written.

Later on, given the defined meta-models, the relations between them can be written. So for example, in this phase the inherent relation between the *UML Class Diagram* meta-model and the *Java Code* meta-model will be written. Analogously, the relations between *Java Code*, *JavaDoc*, *UML Sequence Diagram* and other meta-models of the Java technological space are also to be defined. All of these relations are developed during the work of this thesis.

After having this network of meta-models ready, creating an actual network of concrete models of any software system is trivial, thus a showcase using a synchronization method from the current academic literature is applied to illustrate the synchronization between some representative meta-models.

It is out of the scope of this work a creation or implementation of a synchronization algorithm, as well as a theoretic analysis of the problem or the analysis of performance or completeness of the relations.

2 BIBLIOGRAPHIC REVIEW

Before describing the development of this thesis is important to review the academic state-of-art of model synchronization as well as to review some important definitions regarded to Model-driven Engineering.

2.1 Important Definitions

Below is a list of necessary basic concepts definition, that will be used throughout this document. Some of these definitions are rather narrower than they could be, but for the scope of this thesis they seem to be suitable.

Technological Space: According to the definition from Kurtev et al. (2002, p. 1): *"A technological space is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities. It is often associated to a given user community with shared know-how, educational support, common literature and even workshop and conference meetings"*. By *Java Technological Space* is meant the set of commonly used models, practices, techniques and technologies in Java software development. For example, object-oriented development, unit tests, code documentation and the *Java Virtual Machine* are items of the Java technological space.

System: The term system is used interchangeably with the terms software, or program. And is intended to represent a software unit that abstracts a reality (or problem) and is abstracted by one or more models.

Model: The definition for software model used is given by Seidewitz (2003, p. 27): *"A model is a set of statements about some system under study (SUS). Here, statement means some expression about the SUS that can be considered true or false"*. According to them, models can be used (1) to describe a system, in this case the model makes statements about the SUS, an example is an *UML sequence diagram* employed to help understand the behavior of a Java system. But models can also be used (2) to specify a system, in this case it is used in the validation of the system, an example is a *UML class diagram* employed as design specification of a Java system. Further examples of models, according to this definition, are a *relational database diagram* of a database, the documentation of a system in *Java Doc* or even a Java source-code. The term *artifact* is used as a synonym for model here.

Modeling Language: If, according to definition of model, a model is a set of statements, one needs a way to express such statements. Therefore *"A modeling language lets us express the statements in models of some class of SUS"* (Seidewitz, 2003, p. 28). Examples of modeling languages are the *UML*, the *diagram notation for relational database diagram* or the Java language.

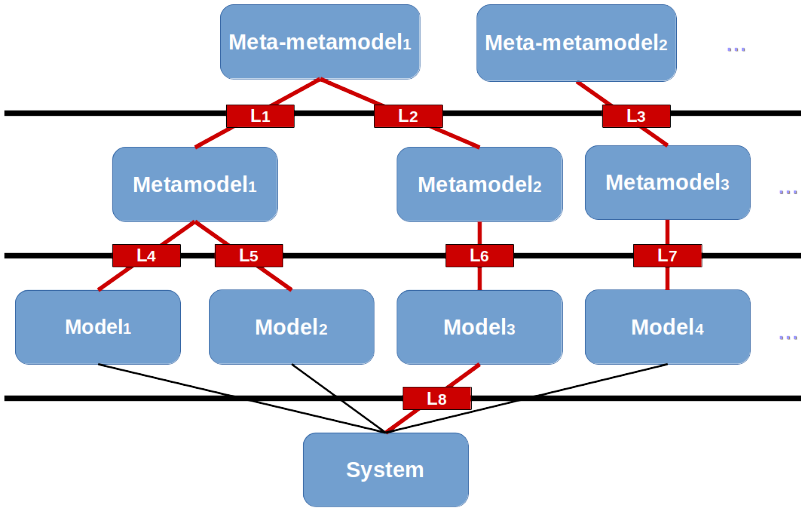
Meta-model: Seidewitz (2003, p. 28) affirms also: *"A metamodel is a specification model for a class of SUS where each SUS in the class is itself a valid model expressed in a certain modeling language"*, that means, a meta-model is a model used to specify another model. Furthermore, *"a metamodel makes statements about what can be expressed in the valid models of a certain modeling language"* (Seidewitz, 2003, p. 28). In other words, a meta-model specifies what can be written using a certain modeling language. Thus, examples of meta-models are *UML specification document* (OMG, 2007), the *entity-relationship meta-model* (Chen, 1976) or the Java meta-model (one example is to be found in Heidenreich et al. (2009)). Finally Seidewitz (2003, p. 29) also claims: *"Because a metamodel is a model, we express it in some modeling language"*. One example of a modeling language for meta-models is the *EMF Ecore*[1].

Meta-metamodel: Analogously to the meta-model definition, *"a meta-metamodel is a specification model of a class of SUS where each SUS is itself a valid meta-model expressed in a certain meta-modeling language"*. To put in other words, a meta-metamodel is a model used to specify another meta-model. An example of meta-metamodel is the *MOF*(Omg, 2015). It is to note also, that such derivation can be done iteratively in the sense that a $meta^3model$ definition is also possible, although it is not useful for the scope of this thesis. The figures 2.1 and 2.2 illustrate our understanding of the definitions above.

Model Relation: Model relation here is defined abstractly as every relationship or constraint possible to happen between one source model and one target model. For instance, the models *UML class diagram* and Java code have a relation, because once a new class is created in the class diagram, the correspondent class has to be created in the Java code. Moreover, a *UML class diagram* with contracts definitions (pre and post-conditions) have a relation to the *JUnit model*, once that the formers have to be tested correspondingly in the latter.

[1] https://eclipse.org/modeling/emf

Figure 2.1: Depiction of the theoretical definitions of system, model, metamodel, meta-metamodel and modeling language. Like stated before, a system is abstracted by models, which themselves are abstracted by meta-models and so forth. A model used to specify (see the model definition) has a red line binding it and its SUS labeled with the identifier of the respective modeling language.

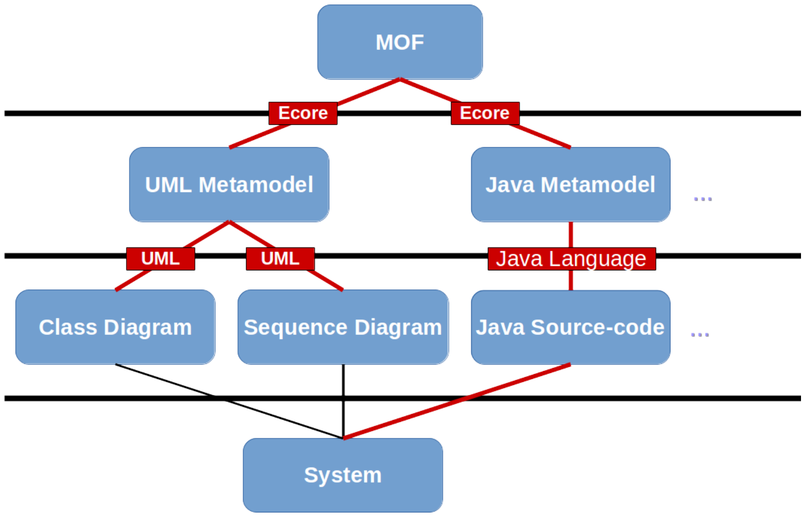


Source: The author

Model Transformation: Model transformation can be viewed as common data transformation – very common in computer science – with the specificity of dealing with models Czarnecki and Helsen (2006). More specifically, model transformation is defined here as a function $t : M \rightarrow N$, where $t(m) = n$ means that a target model $n \in N$ is created from a source model $m \in M$, M and N being respectively the set of all valid models of the meta-models Φ_M and Φ_N. Practical example: Creation of Java code from *UML class diagram*. Note that, model transformation is by nature deterministic, unidirectional and does not preserve the information of the target model (e.g. comments in the Java code).

Model Synchronization: The goal of model synchronization is to maintain all relations between the models of a system consistent/correct as updates are performed over them Diskin (2011). More specifically, model synchronization is defined here as a function $s : MxMxNxN \rightarrow MxN$, where $s(m_0, m_1, n_0, n_1) = (m_2, n_2)$ means that final synchronized models m_2 and n_2 are created from the initial synchronized models m_0 and n_o and the modified non-synchronized models m_1 and n_1. Practical example: Modification of a method name in the *UML class diagram* has to be forwarded to the Java code, without losing extra information of it (e.g. comments). Note that, model synchronization is deterministic, bidirectional and preserves the informations of the both models. Other terms for model synchronization used interchangeably throughout this document are iterative

Figure 2.2: A more concrete and practical illustration of the definitions of system, model, metamodel, meta-metamodel and modeling language. This example shows a scenario very close to the implementation made in chapter 3.

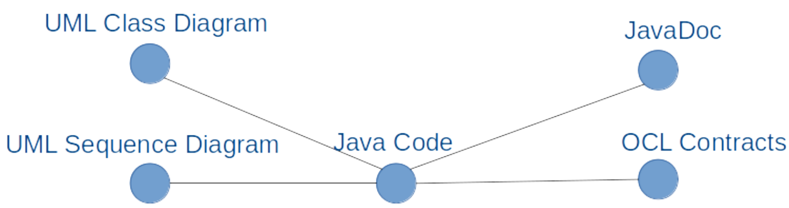


Source: The author

or information preserving bidirectional model transformation.

Network of Models: A network of models of a system S is an undirected graph $G = (V, E)$, whereas each vertex $v_i \in V$ represents a unique model i abstracting S, and an edge (v_i, v_j) exists if, and only if there is a relation defined between both models i and j. In the figure 2.3 is an example of a network of models, illustrating the possible complexity of such network. More discussion is to find in Mens and Van Gorp (2006).

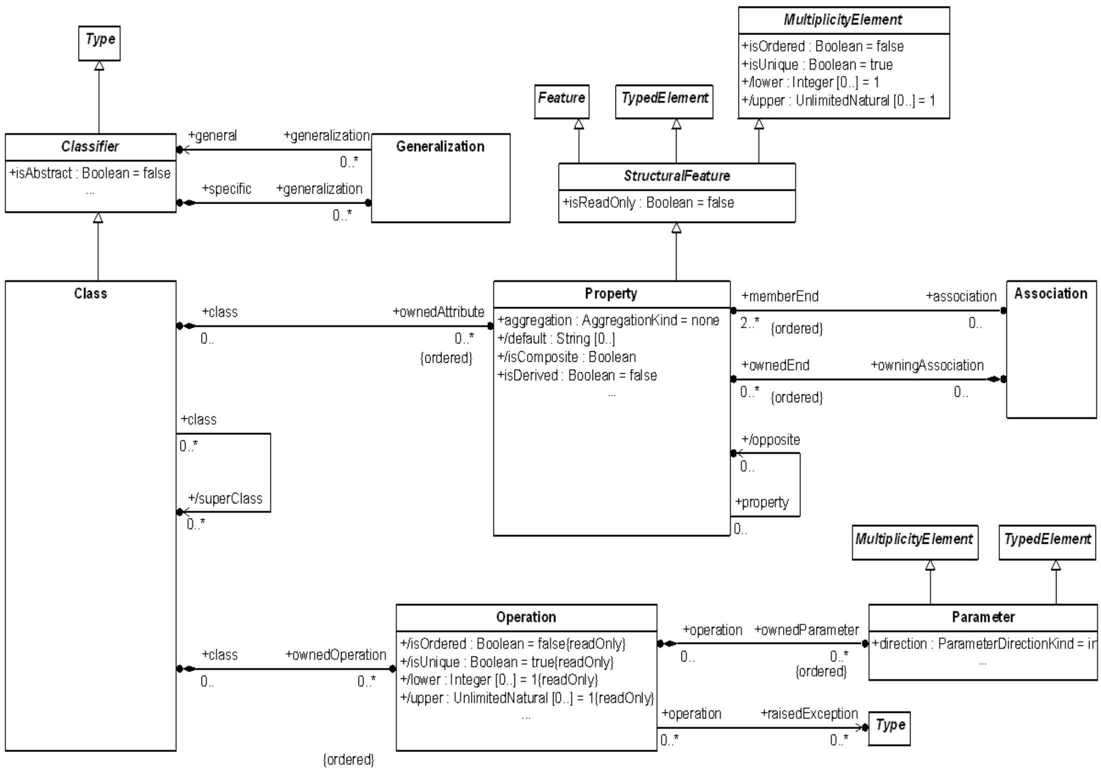
Figure 2.3: An example of a network of models very similar to the one developed in this work.



Source: The Author

Meta Object Facility: *"The Meta Object Facility (MOF) provides an open and platform-independent metadata management framework and associated set of metadata services to enable the development and interoperability of model and metadata driven systems"* (Omg, 2015). The MOF describes therefore the MOF modeling language, which is used to model the meta-metamodel utilized in this thesis. Essentially, it inherits much from the UML and deals with the ideas of classes, properties and associations, providing and extensible but simple fashion to define meta-models. The figure 2.4 shows the essential part of MOF.

Figure 2.4: Essential MOF definition, which handles basically classes, properties, operations, associations, and generalization.

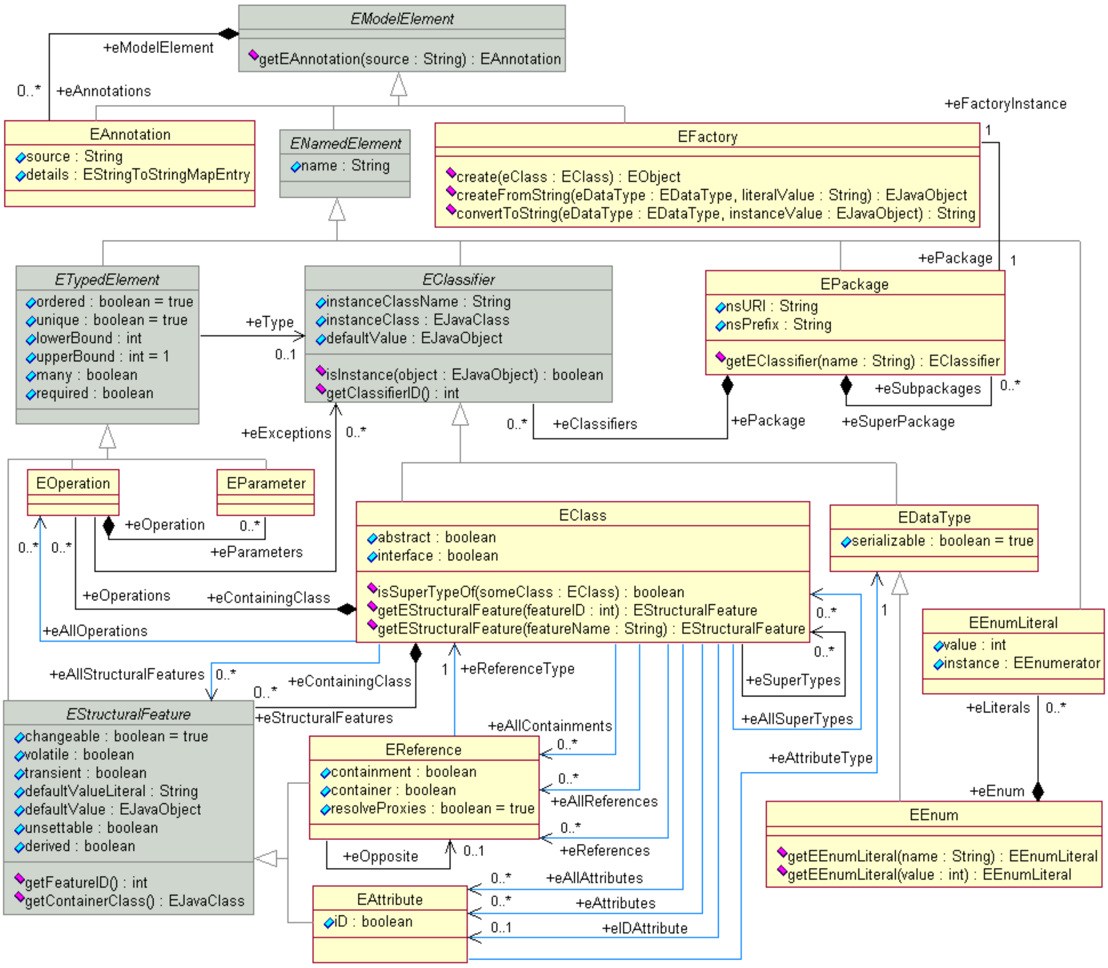


Source: (Omg, 2015, p. 27)

Ecore: Ecore[2] is the meta-metamodel utilized in this thesis to describe all the applied meta-models (e.g the Java meta-model). Ecore is an initiative of the EMF Project and aims to provide not only a meta-metamodel but a set of tools for criating meta-models, like an Eclipse plug-in generation feature, that enables the model developer to easily test and debug its meta-models. The Ecore meta-metamodel is at least similar to the essential MOF standard, and that is the reasons it is applied here. A proof of such compliance is not know by us though. The figure 2.5 shows the essential part of Ecore.

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

Figure 2.5: Ecore definition illustrating the use of classes, attributes, operations, references and super types, analogously to the figure 2.4



Source: http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html. On the December 29th, 2015

Triple Graph: With the use of a triple graph a relation between a source model S and a target model T are abstracted into a triple (G_s, G_c, G_t) – where G_s is the graph representation of source model elements, G_t is the graph representation of target model elements, and G_c represents the correspondence between the two set of model elements – together with two mappings $s_g : G_c \to G_s$ and $t_g : G_c \to G_t$, which bind the three graphs (Hermann et al., 2011).

In this case, a modification in the triple graph $G = (G_s, G_c, G_t)$, that leads to a new triple graph $H = (H_s, H_c, H_t)$ consists in a triple graph morphism $m : G \to H$, with $m = (m_s, m_c, m_t)$. According to the figure 2.6.

Triple Rule: A triple rule is a triple graph morphism $t_r = (s, c, t) : L \to R$, where L and R are called respectively the left-hand the right-hand sides (Ehrig et al., 2007).

Figure 2.6: The morphism $m : G \to H$ is a triple graph $m = (m_s, m_c, m_t)$.

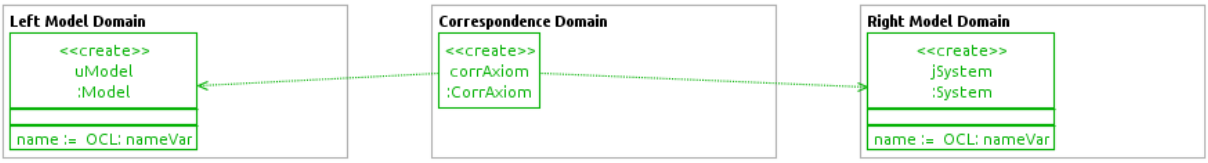
$$
\begin{tikzcd}
G = (G^S \arrow[d, "m^S"'] & G^C \arrow[l, "s_G"'] \arrow[r, "t_G"] \arrow[d, "m^C"'] & G^T) \arrow[d, "m^T"'] \\
H = (H^S & H^C \arrow[l, "s_H"] \arrow[r, "t_H"'] & H^T)
\end{tikzcd}
$$

Source: (Hermann et al., 2011)

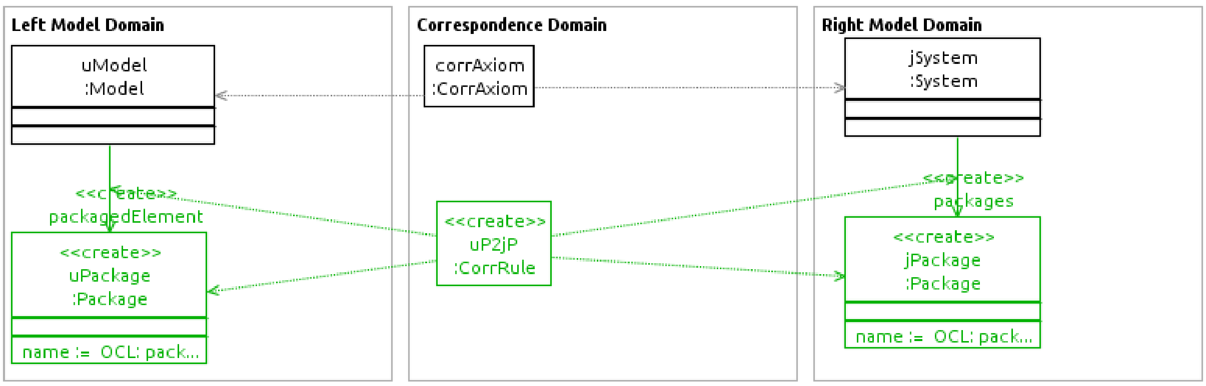
Triple Axiom: A triple axiom is a triple rule $t_a = (s, c, t) : \emptyset \to R$. In order to apply such definitions in the practice, it is common to use attributed graphs and a easier to read diagram scheme depicted in the figure 2.7.

Figure 2.7: In this kind of diagram for triple rules a triple graph is represented by three columns (left mode domain, correspondence domain, and right model domain) each one representing respectively the source model elements, the coorrespondence between source and target and finally the target model elements. A triple rule in turn is represented by a triple graph in black and a triple graph in green, respectively the left-hand and right-hand side (see 2.7b). As an axiom is a triple rule with empty left-hand side only green graph occurs (see 2.7a)

(a) Triple axiom example for the relation between UML and Java



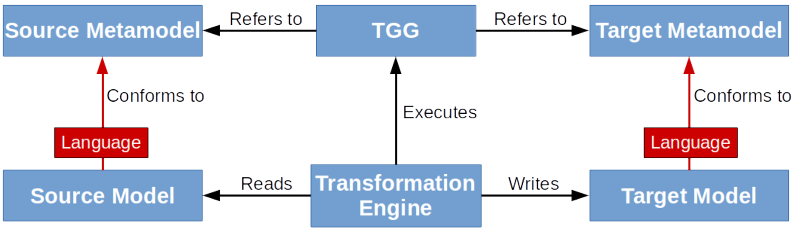
(b) Triple Rule example for the relation between UML package and Java package



Source: The author

Triple Graph Grammar: A triple graph grammar $TGG = (t_a, T_{rules})$ consists of a triple axiom t_a and a set of triple rules T_{rules} Giese et al. (2010a, p. 4). While triple graphs are used as a description of the relations between two meta-models, TGG's describe the semantics of the transformation procedure, where triple rules correspond to operational rules of the formal semantic (see chapter 4). Figure 2.8 summarizes the so far defined terms.

Figure 2.8: Illustration of the definitions model relation, transformation and synchronization as well as triple graph grammars (TGG). Relations between metamodels are coded by triple grammars; modifications in the models are coded by triple rules, which are then organized in a TGG. A TGG is interpreted as operational semantic definitions and executed according to the models. Again the concept of modeling language is pictured as red lines.



Source: Adapted from Czarnecki and Helsen (2006, p. 623)

2.2 Related Work

A work similar to the one of this thesis, which builds a network of meta-models of the Java technogical space, by describing them and their intrinsic relations is not known by us. In this sense the results of this thesis is novel and contributive. Nevertheless some endeavors have been made in order to code relations between some meta-models and mainly to develop theoretical results and synchronization methods. Heidenreich et al. present in (2009) and (2010) a Java meta-model using *Ecore*, what influenced considerably the development of our work, although it has not been used by us because of its size and unnecessary comprehensibility for our needs. Greenyer et al. (2008) comes up with a transformation between *UML activity diagrams* and *CSP diagrams* using *TGG*. Foss et al. (2011) defined the translation between *UML* and *Simulink* using graph grammars. Blouin et al. (2014) reports about the synchronization between some specific meta-models of the automotive standards and influences our work, by using the same modeling language and synchronization method as us, namely *EMF* (Steinberg et al., 2008) and *MoTE* (Giese et al., 2010a). Finally Giese et al. (2010b) introduce their approach to the synchronization of two automotive industry meta-models, lightening in the paper the *MoTE* tool and its algorithm for synchronization.

We judge that the *MoTE* tool is the most adequate option for our needs, once literature about it is widely available (see also (Giese and Hildebrandt, 2009) and (Hildebrandt et al., 2012)). Nevertheless there are other attempts to build a model synchronization tool, like the *ATL Eclipse Plug-in* (Jouault et al., 2008), which uses the *Atlas Transformation Language* to code the relations between models; the Medini QVT [3], which claims to implement the *Query/View/Transformation Language* to code the relations; and the FUJABA (Nickel et al., 2000), in which relations are coded using *Triple Graph Grammars*. Hildebrandt et al. (2013) also publicized a survey on synchronization tools based on TGG. Other publications aim to solve specific problems, like the ones in Hermann et al. (2011), Xiong et al. (2007), Giese and Wagner (2006), Ivkovic and Kontogiannis (2004), or Song et al. (2011), where advanced algorithms for bidirectional synchronization have been proposed.

A research road-map for model synchronization found in France and Rumpe (2007) gives an overview on the realm, and together with Mattsson et al. (2009) show an interesting point of view about the challenges. Seidewitz (2003) writes an interesting reflection about what models mean and how to interpret them and in Mens and Van Gorp (2006) a taxonomy for model transformation is proposed, what helps to carry more precise analysis. In Czarnecki and Helsen (2006) a survey was driven and a framework for classification of model transformation approaches was presented. In Diskin et al. (2014) and (2016) a taxonomy for a network of models is presented and in Diskin (2011) a theoretical algebraic basis is proposed.

Additionally, one can judge by the date of publication of these works, that the topic of model synchronization is extremely active and is actually the edge of current academic research, what motivates even more the development of this thesis.

[3]http://projects.ikv.de/qvt

3 META-MODEL RELATIONS IN THE JAVA TECHNOLOGICAL SPACE

With the terms and the theoretical basis clarified the report of the main development phase of the thesis is shown below. The goal here is to present the developed meta-model definitions and the relations between them, as well as justify the choices made during the work.

3.1 Meta-models

The modeling language used to write these meta-models is the *EMF Ecore* and the tool used is the special version for model development of the *Eclipse Mars 4.5.1 IDE*[1], which eases the creations of models ans their diagrams and the generation of plug-ins necessary for running the transformations, thus being more suitable than the alternatives of the *Netbeans IDE*[2] or the FUJABA (Nickel et al., 2000), whose support or popularity in the community are not so valuable. The *Ecore* language has been chosen not only for its extensible documentation and popularity in the community, but also for its ease to use in the *Eclipse IDE*. The meta-models are listed in the sections below.

3.1.1 UML

The meta-model utilized for UML represents the version 2.0 of the standard and is provided by the *EMF plug-in*[3] for Eclipse, which clearly integrates easily with the IDE and seems to be suitable for our needs. Alternatively we could use the meta-model provided by the OMG, but then unnecessary work of adaption could late our progress. A meta-model for the latest version of the UML (2.5) has not been found. The figures **??**, 3.2 and 3.3 address simplified views of the UML meta-model for respectively the classifiers, interactions and contracts concerns.

[1] https://projects.eclipse.org/releases/mars
[2] https://eclipse.org
[3] https://eclipse.org/modeling/emf

3.1.2 Java

For the Java a brand new meta-model was designed in regard to the necessities of this work. Other possibilities included the meta-model[4] provided by the *Eclipse IDE*, whose simplicity hindered its use, or yet the one found in Heidenreich et al. (2010), but this one happens to be so extensive that could bring unnecessary complexity. The figure 3.4 reports the created Java meta-model.

3.2 Relations

With all the meta-models defined, the definition of the relations between their elements can be done. In order to accomplish that, TGG is used to code the relations, because it has been extensively used in current academic research; and is supported by several tools for synchronization. Other options included the *ATL* (Jouault et al., 2008), which does not seem to be enough ripe for our use; or the *Henshin* (Arendt et al., 2010), that is not widely supported for the best tools (Hildebrandt et al., 2013). A theoretical basis of TGG has been given in the section 2.1. Here the relations written by us are presented as well as some explanations over them.

3.2.1 Splitting the Meta-models

Once the meta-models might be relatively big and comprise a huge number of elements, it is interesting to split them into smaller pieces for a specific set of relations. Take for example the UML, that includes a large number of different concerns (e.g. classifiers, state machines, activities, interaction, etc) and thus may be split into sub meta-models in order to ease the writing of the relations. These relations are given in the following sections. A later joining of all diagrams should not be problematic.

3.2.2 UmlClass2Java

EXPLAIN FIGURE 3.5 to 3.6

3.2.3 UmlInteraction2Java

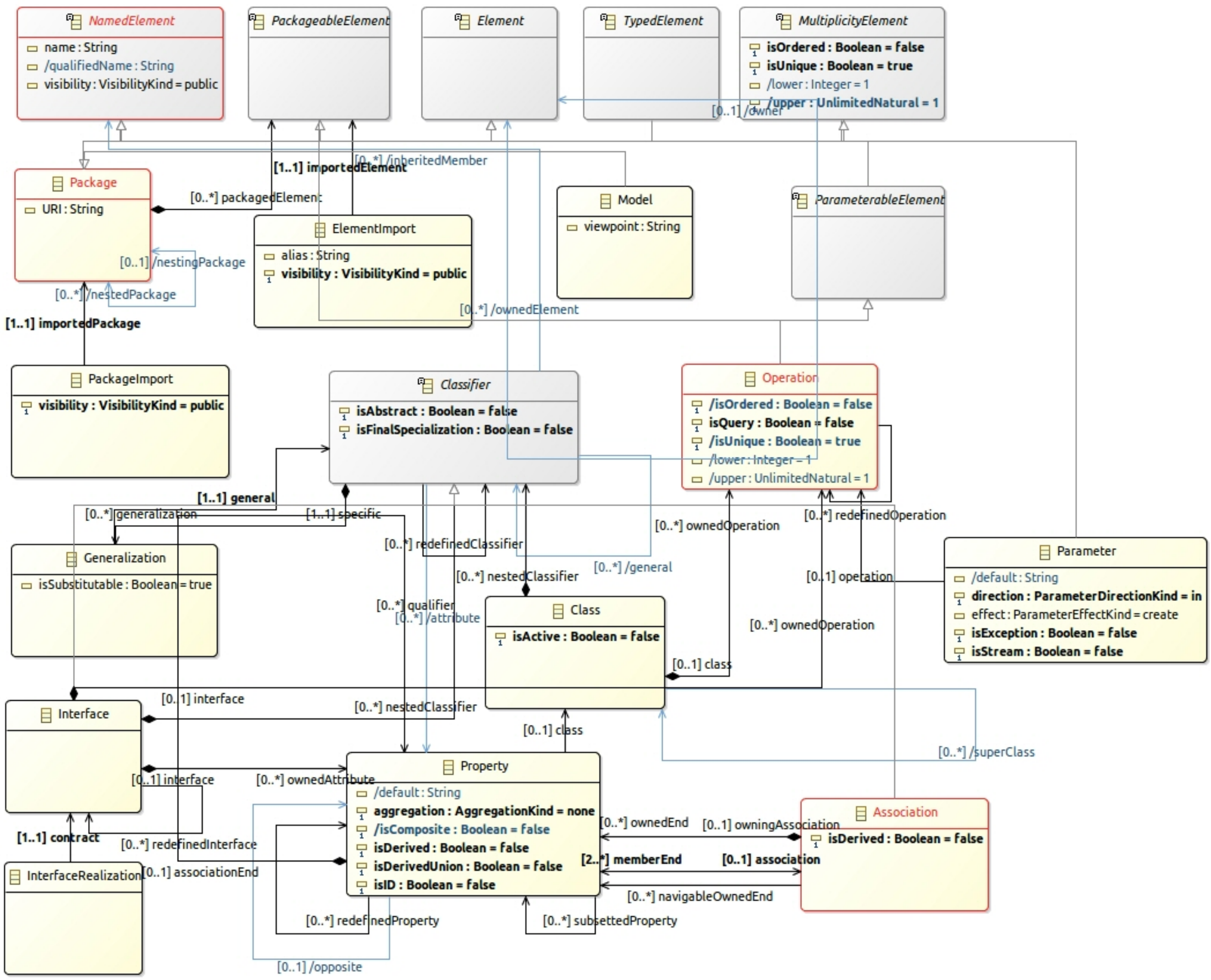
FIGURE 3.7 to 3.9

3.2.4 UmlContracts2Java

FIGURE 3.10 to 3.14

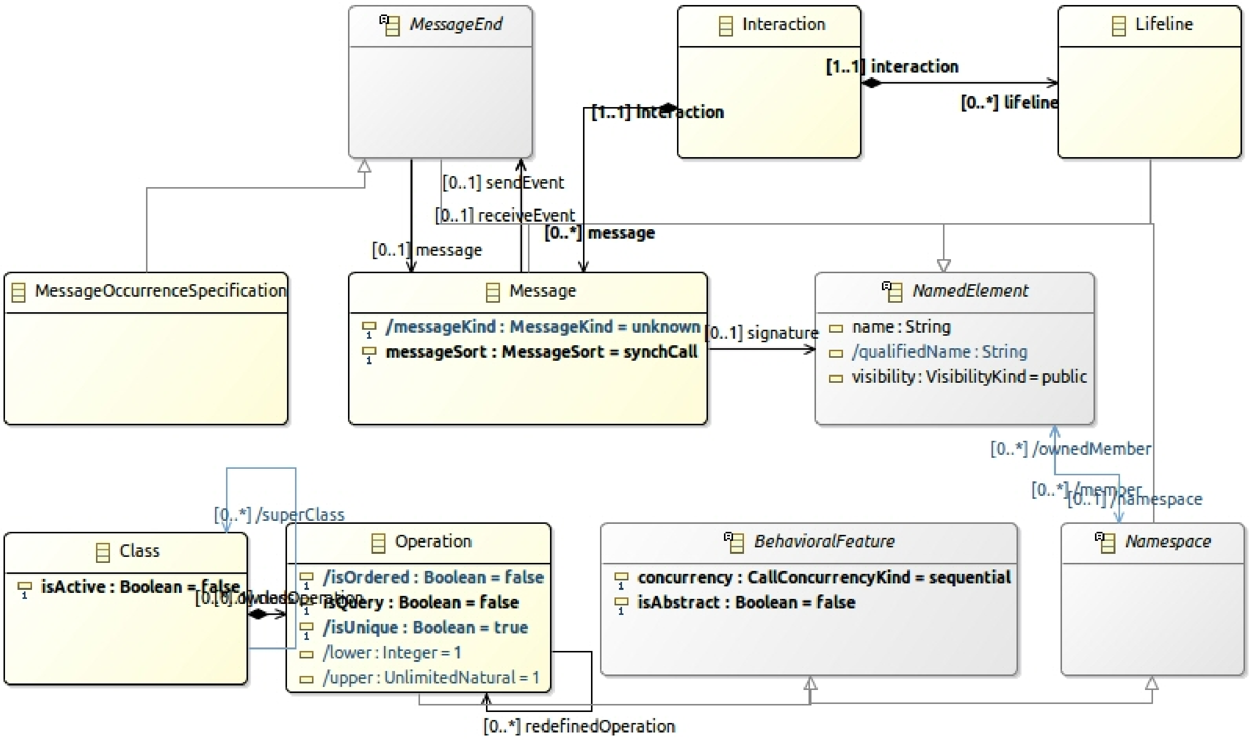
[4]http://www.eclipse.org/modeling/emf/downloads

Figure 3.1: A simplified view of the UML meta-model focused on the elements used in classes diagrams. Elements in blue are abstract elements, whilst elements in yellow are concrete. Some features like operations and some relations between elements were omitted for a better visualization.



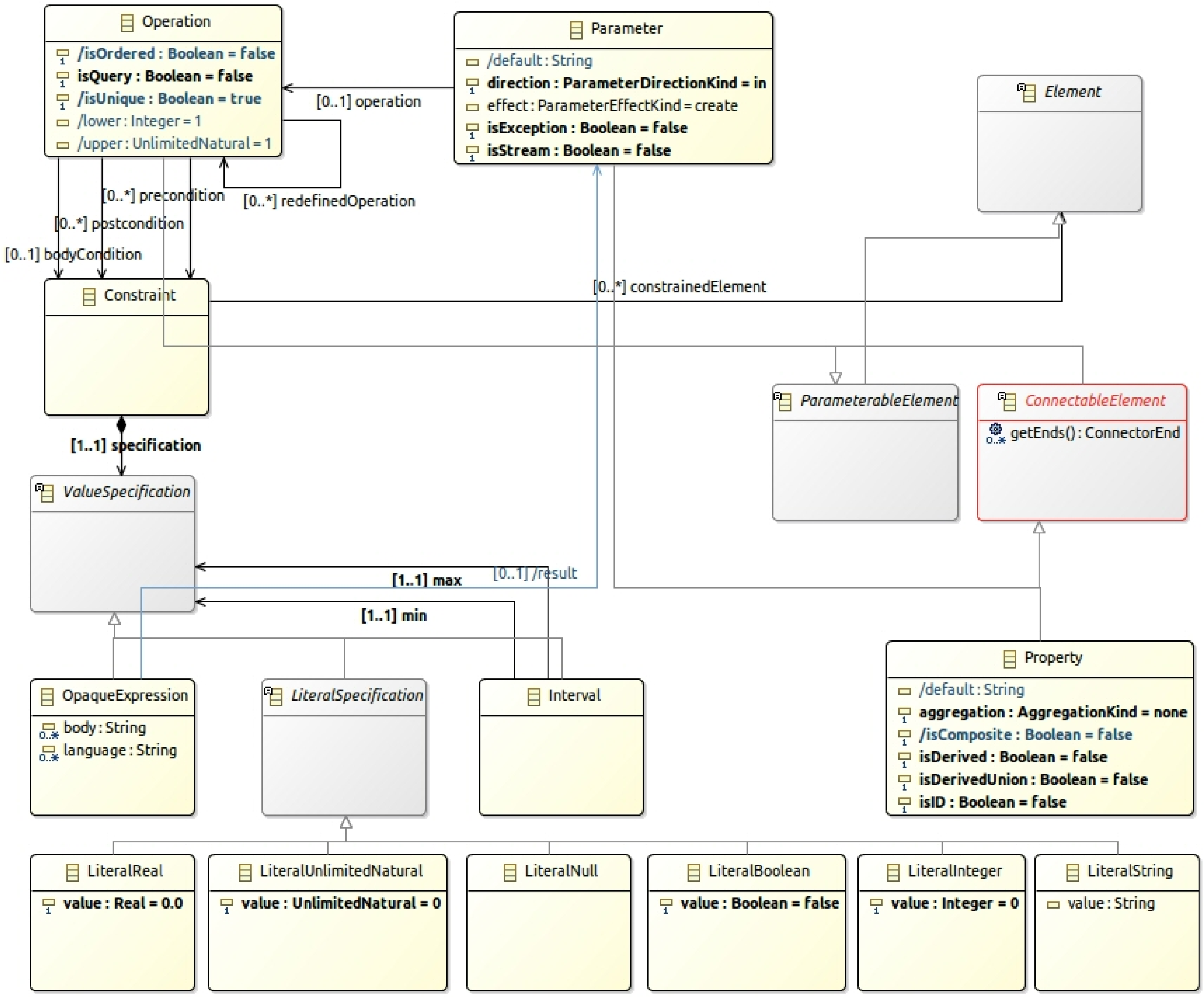
Source: Image created by the author using the *Eclipse IDE*. Meta-model from *EMF plug-in*

Figure 3.2: A simplified view of the UML meta-model focused on the elements used in sequence diagrams.



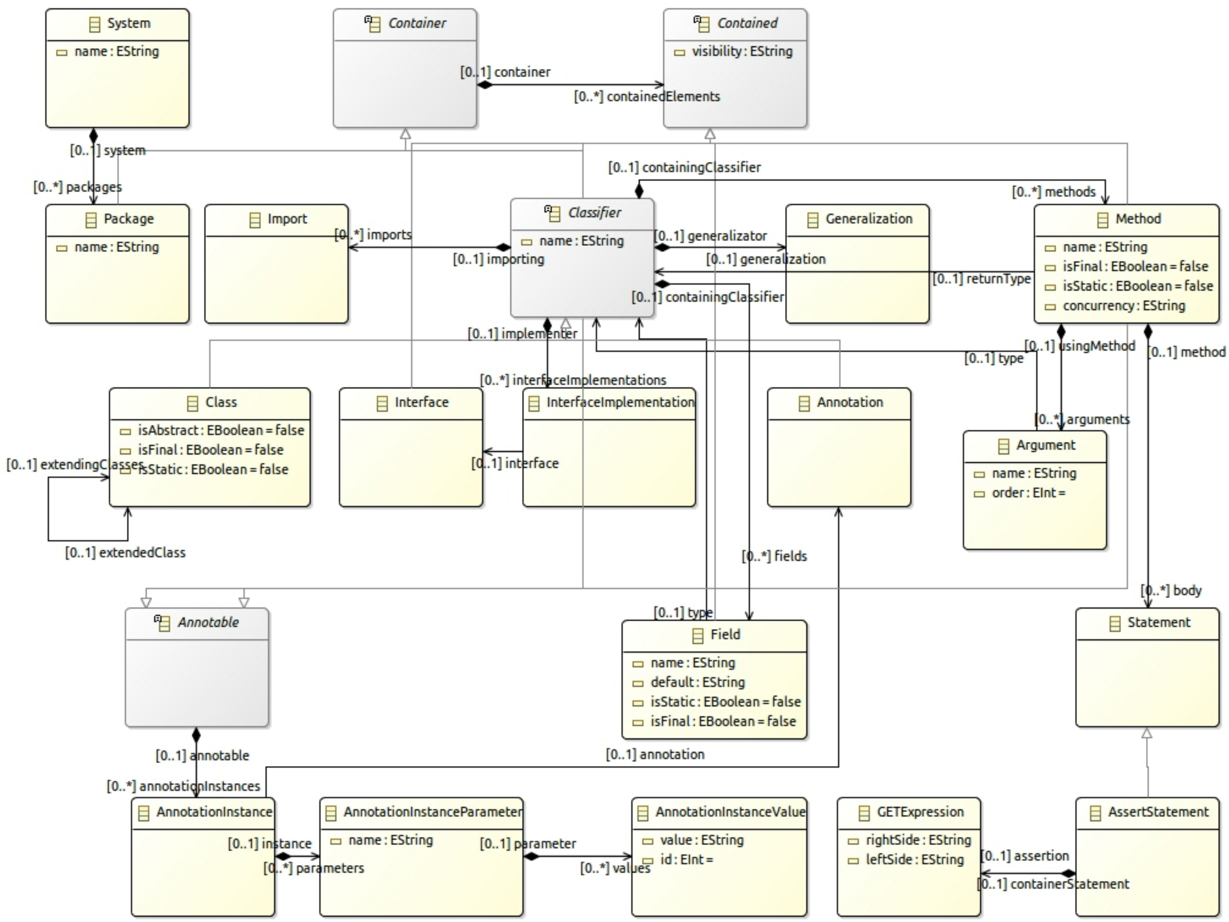
Source: Image created by the author using the *Eclipse IDE*. Meta-model from *EMF plug-in*

Figure 3.3: A simplified view of the UML meta-model focused on the elements used in contracts.



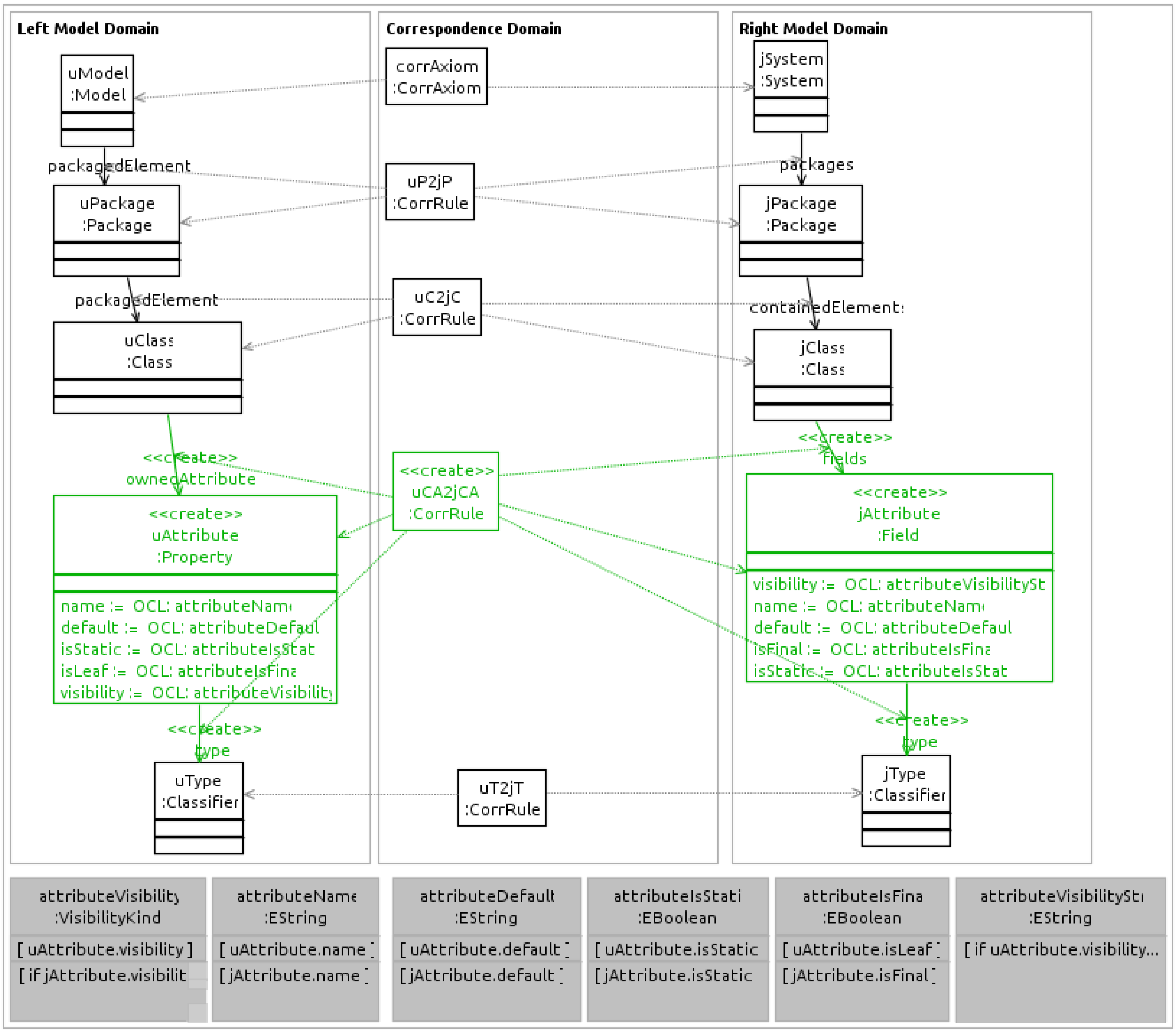
Source: Image created by the author using the *Eclipse IDE*. Meta-model from *EMF plug-in*

Figure 3.4: The Java meta-model created. It goes beyond the common classes and attributes scenario, by comprising also annotations and statements.



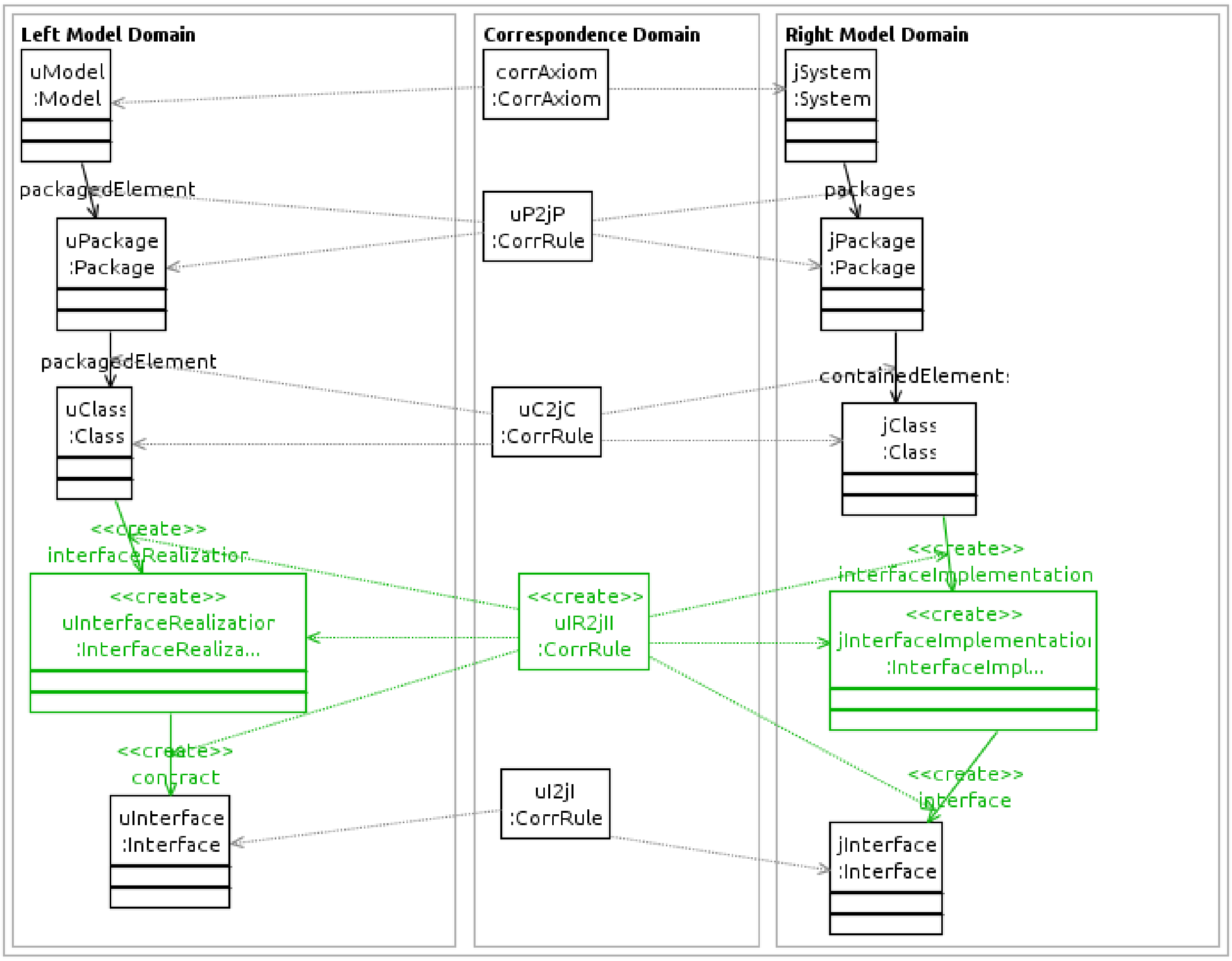
Source: Image created by the author using the *Eclipse IDE*.

Figure 3.5



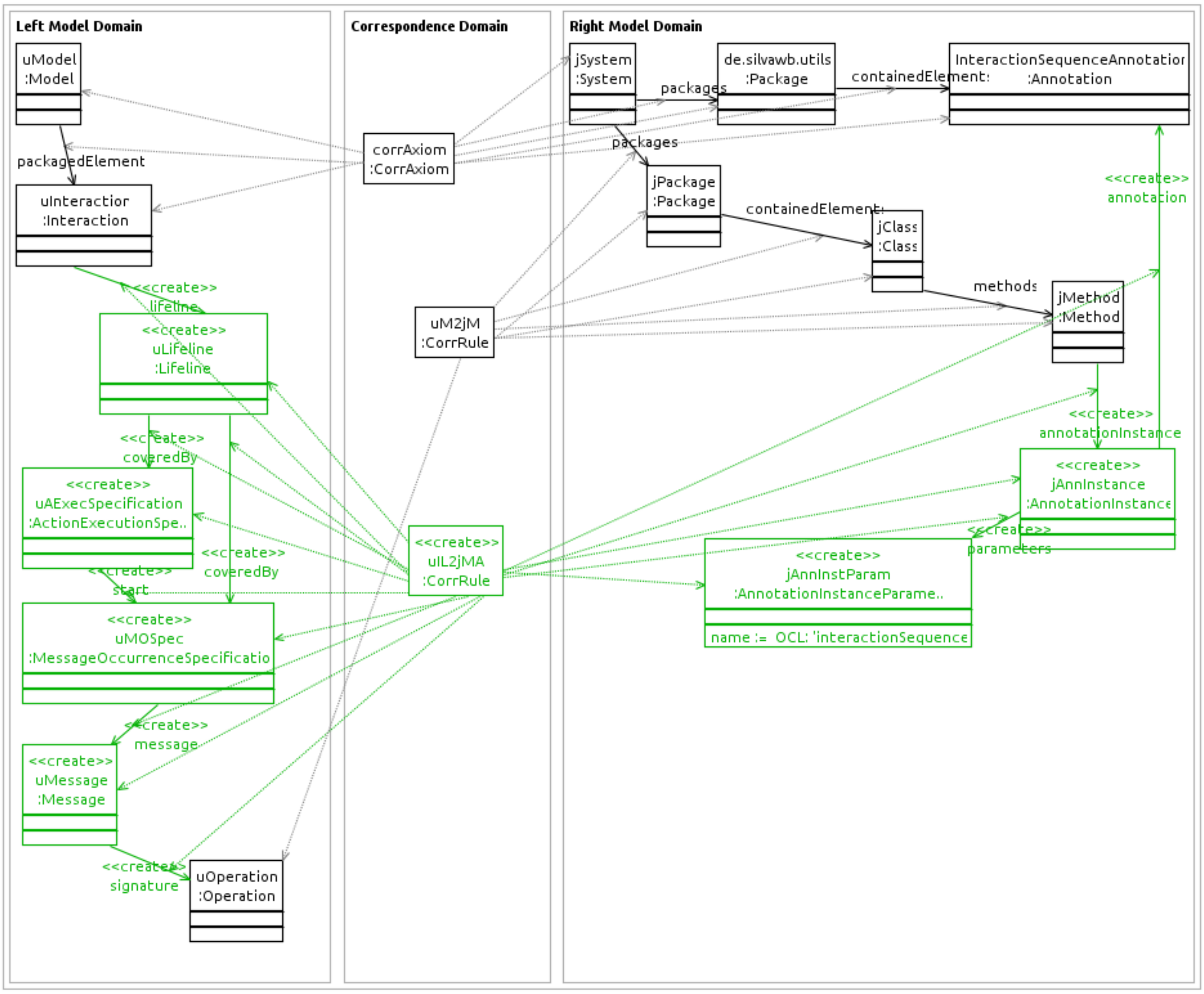
Source: Image created by the author using the *Eclipse IDE*.

Figure 3.6



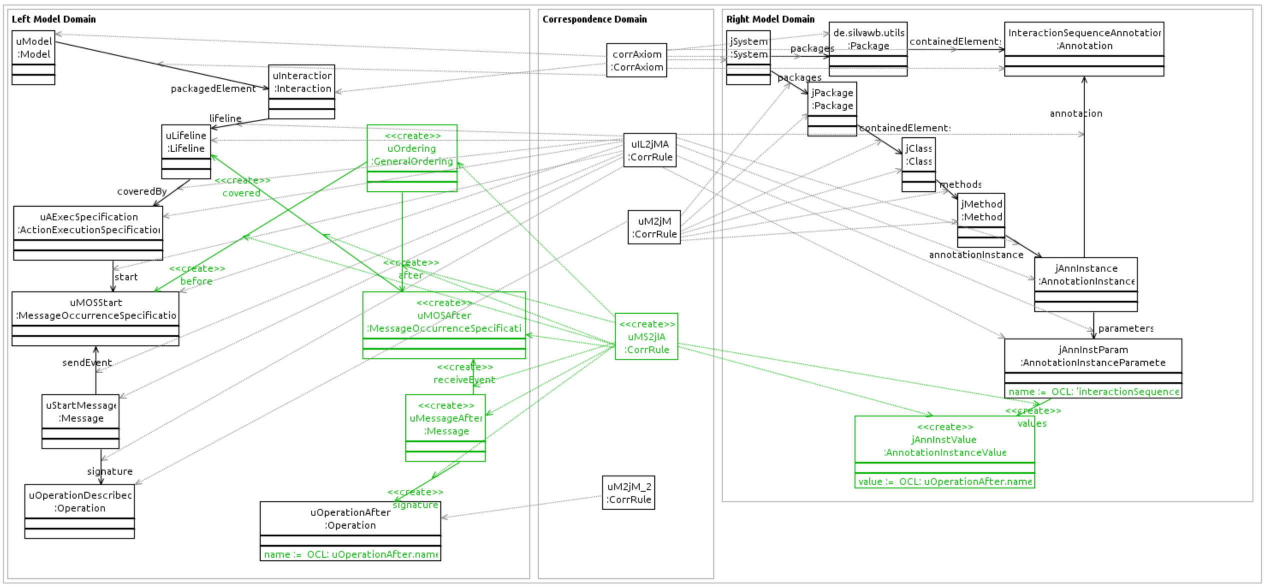
Source: Image created by the author using the *Eclipse IDE*.

Figure 3.7



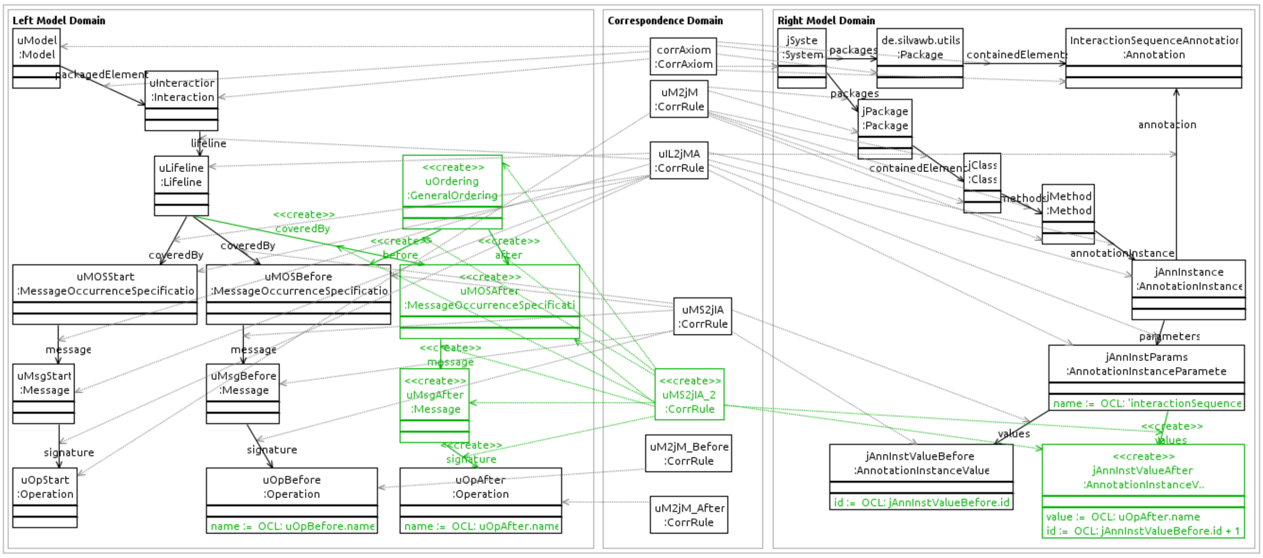
Source: Image created by the author using the *Eclipse IDE*.

Figure 3.8



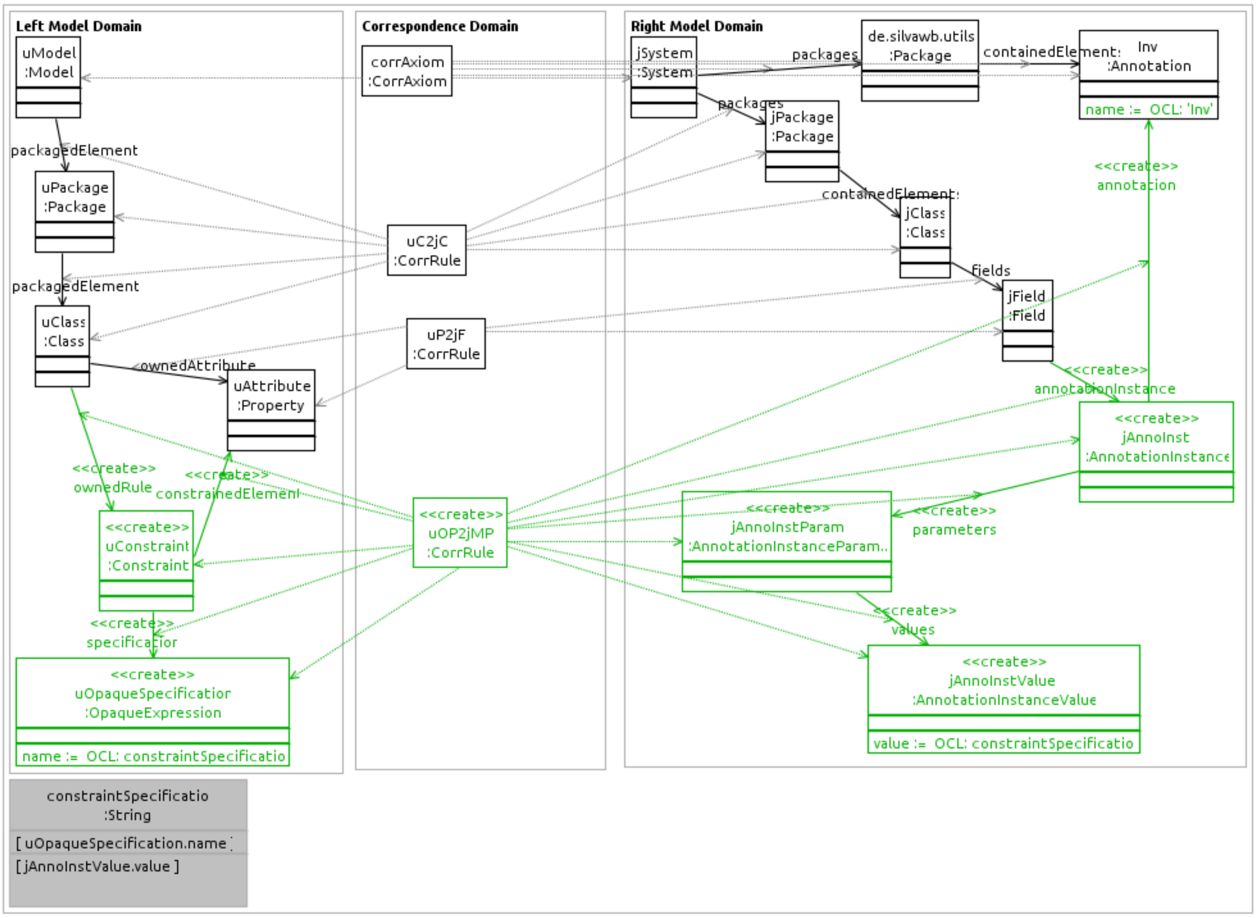
Source: Image created by the author using the *Eclipse IDE*.

Figure 3.9



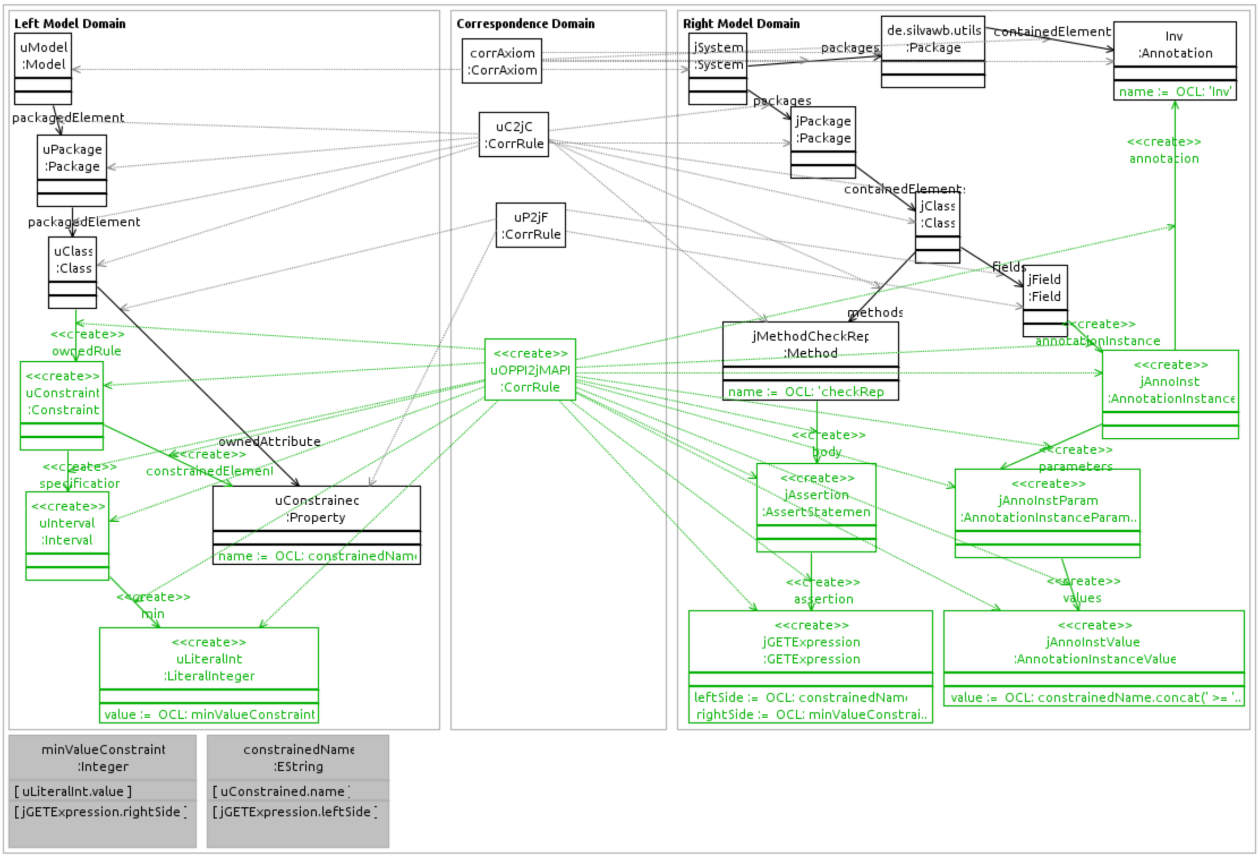
Source: Image created by the author using the *Eclipse IDE*.

Figure 3.10



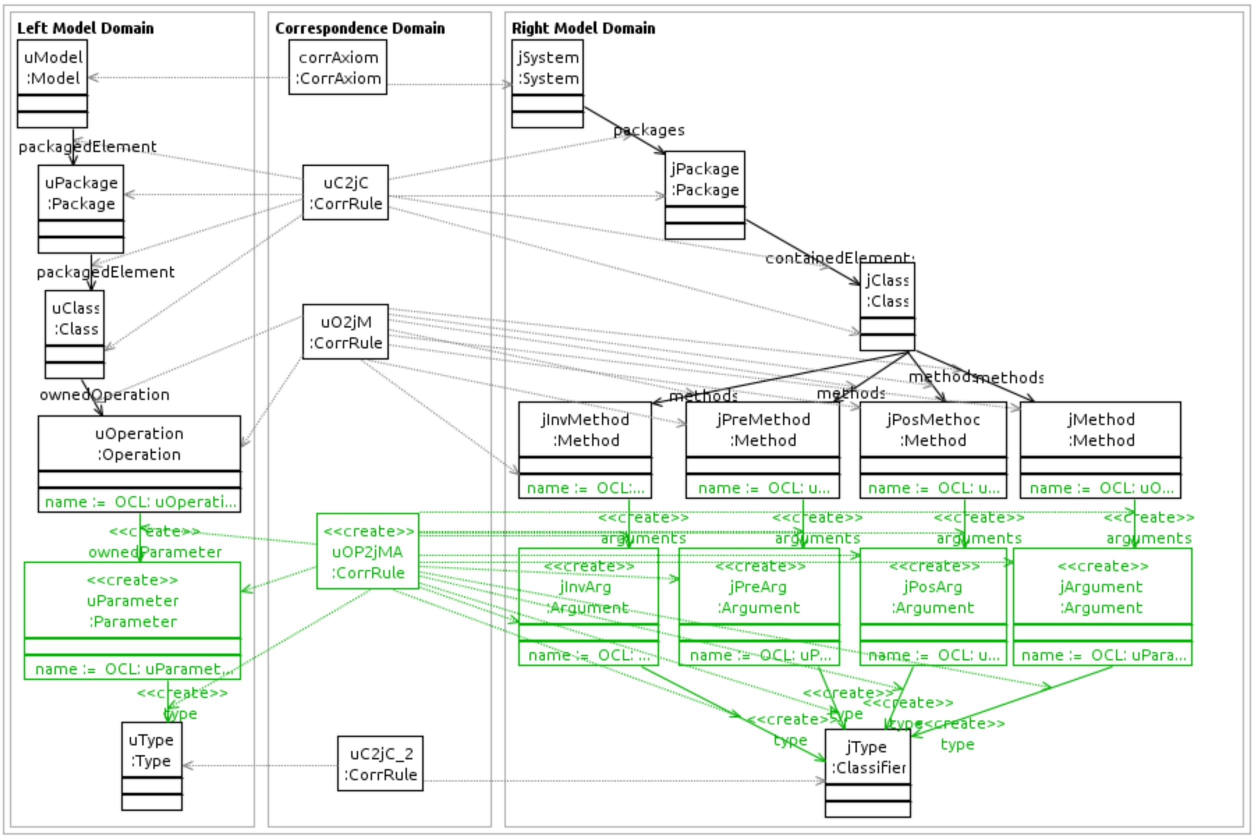
Source: Image created by the author using the *Eclipse IDE*.

Figure 3.11



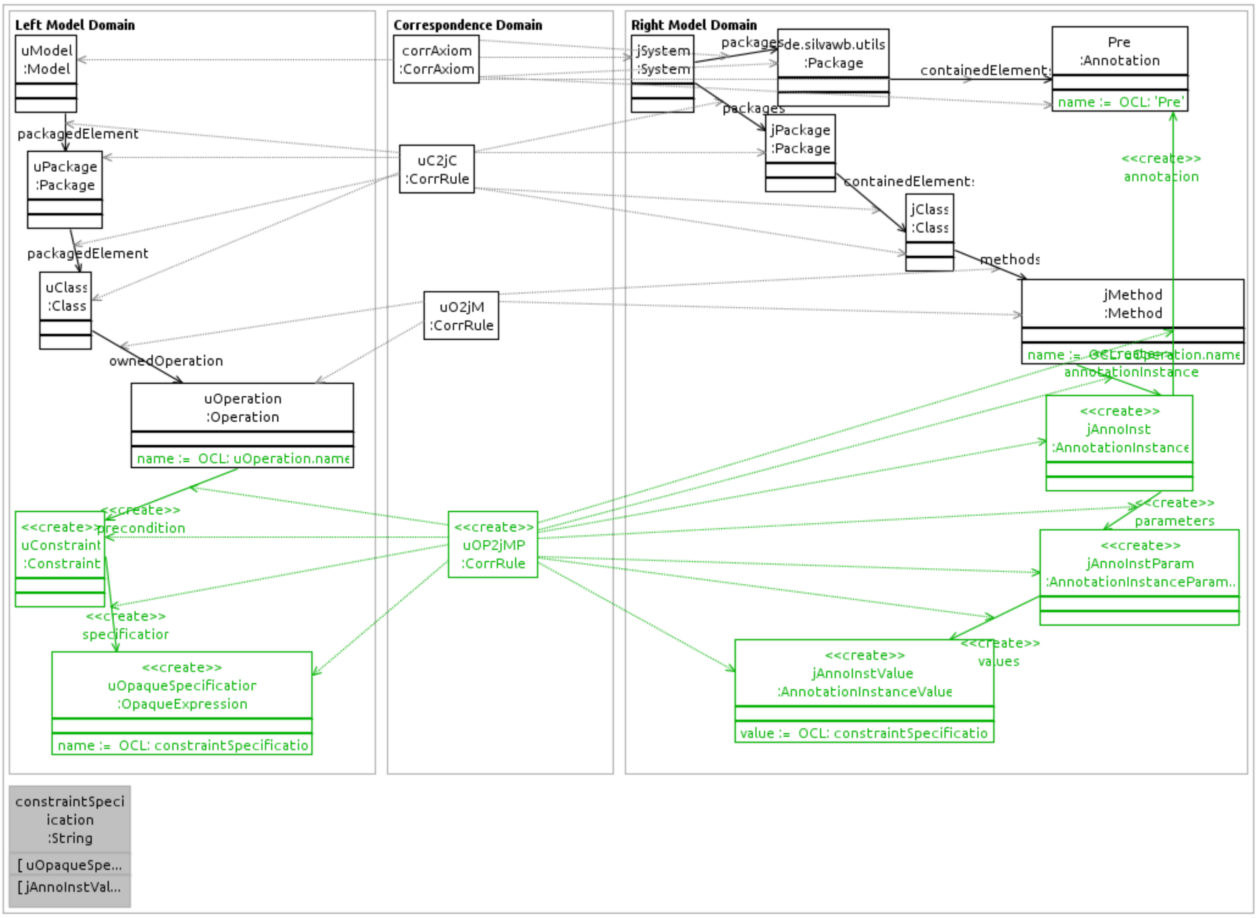
Source: Image created by the author using the *Eclipse IDE*.

Figure 3.12



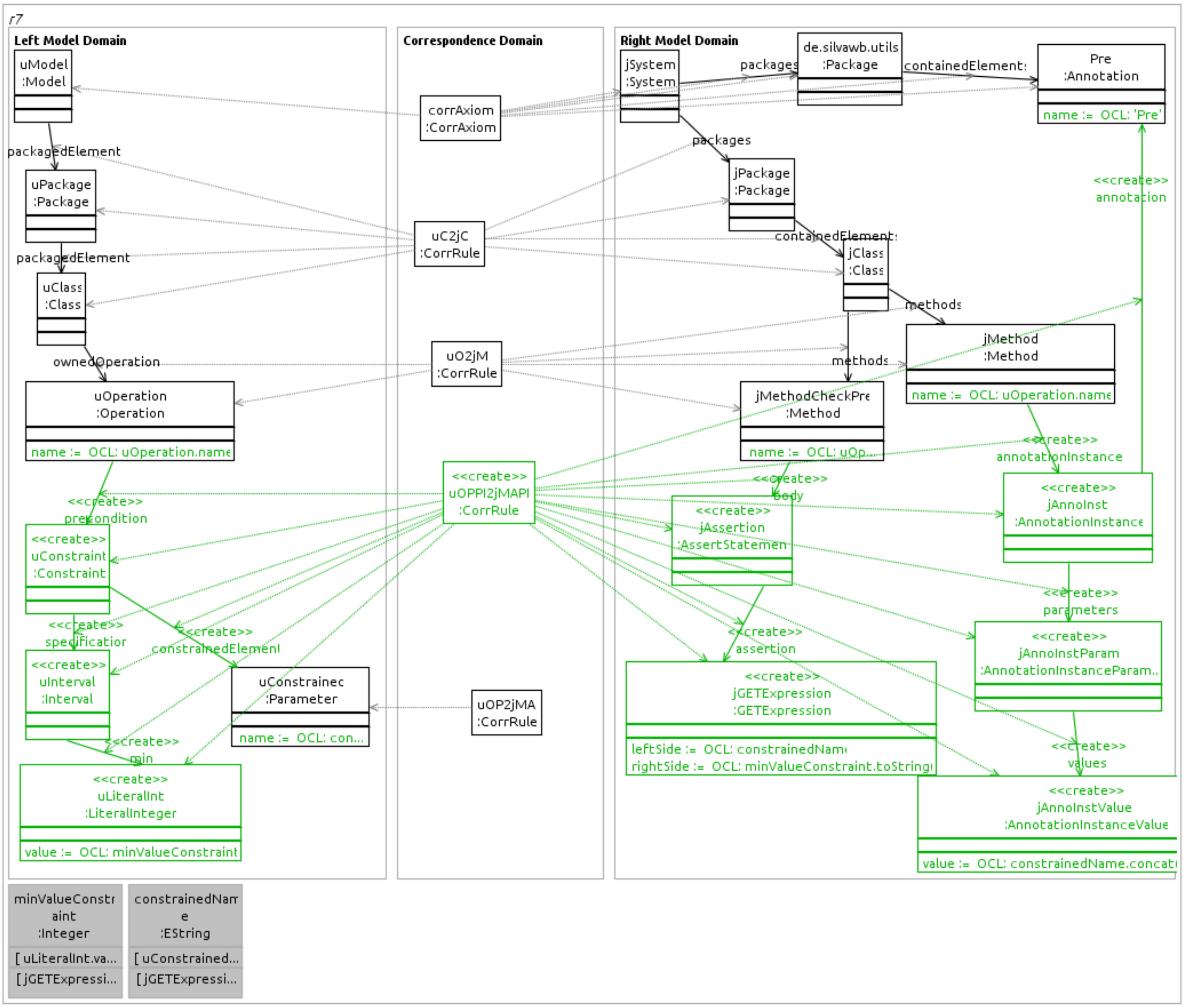
Source: Image created by the author using the *Eclipse IDE*.

Figure 3.13



Source: Image created by the author using the *Eclipse IDE*.

Figure 3.14



Source: Image created by the author using the *Eclipse IDE*.

4 SYNCHRONIZATION BETWEEN SOME MODELS IN THE JAVA TECHNOLOGICAL SPACE

5 CONCLUSION AND DISCUSSION

A network of meta-models of the Java technological space was developed, comprising a set of common meta-models used in Java software and the respective relations between them, as well as synchronization between the models was showcased, demonstrating as final result that the construction of such network and the employment of synchronization has worked and moreover seems to be promising. Despite the outcomes are not complete and not ripe enough to be put in practice, the ideas and the insights reported plus the summary of the state-of-art are valuable for current research and thus contributing.

In the first phase some meta-models were developed (e.g Java), and in the second phase the relations were coded (e.g. umlClass2java). In regard to these both steps the main legacy is the novel relations between some elements of some meta-models (e.g. the relation between UML contracts and Java assertions). The third phase serve to demonstrate briefly the application of synchronization in the practice. Most of the initially set up goals were achieved, even though the relations could rather be more extensive, likewise a more comprehensive synchronization demonstration would be desired.

Some difficulties were found along the work, but they absolutely did not obstructed the success of the final result. The first one the lack of openly available meta-models in the literature or by the vendors. For instance *Oracle* does not publicize any standard meta-model for Java, nor are them easily to find in the literature. One may find alternative versions in the source code of IDE's, but it still requires some cost. Moreover, they are sometimes because their format incompatible with the employed tool, what also hinders the progress of someone's work. The result of this thesis may be a partial solution for that, although meta-models of other technological spaces still lack a similar work. Another complication are the lack of documentation of some tools − in special the *MoTE* −, that makes both the flow of the development and eventual debug tasks sometimes troublesome. *MoTE* might have publications about it and also a good reputation in the community, but a extensive broad documentation of the plug-in for the *Eclipse IDE* is needed. At last, but not least is the performance problem of such tools. Both the *EMF* and *MoTE* need to generate Java code in order to run the synchronization procedure, and with big models this process happens to cost a considerable computation time.

Some points become therefore remarkable for future work. Firstly, an easy to find and accessible tutorial or instructions for the theoretic and practical basis of triple graph grammars is valuable in order to make the use of models synchronization popular among engineers or software developers, who sometimes are not very used to the area and thus express a big rejection to apply such technique in their projects.

Secondly, the work of this thesis can be naturally continued and expanded, by completing the identification of relation between the meta-model or by creating meta-models that satisfy completeness. Not to mention, such relations could be expressed in other languages (e.g. *ATL*) and the same work extended to other technological spaces (e.g. COBOL, C#).

Furthermore, the implementation of a *Eclipse* plug-in should not be a big problem, since the *MoTE* already generates a plug-in for the execution of its algorithm. So the task in this case would be enhance it with a user-friendly interface plus new functionalities and naturally a slightly more practical approach (e.g. handling of actual Java code instead of quasi-theoretic scenarios like Java expressed in XML files).

And lately, a relatively big issue is the use of TGG's for non-MOF-compliant meta-models – or meta-models that are not naturally seen as MOF-compliant, e.g. the complete Java model. Because under certain conditions it could be interesting to see such models from another point of view than the MOF. A clear example is source-code, which can be treated in an easier way with abstract syntax trees. Angyal et al. (2008) suggests a method to treat this case, but anyways it still remains an open problem and certainly a future challenge.

REFERENCES

László Angyal, László Lengyel, and Hassan Charaf. Novel techniques for model-code synchronization. *Electronic Communications of the EASST*, 8, 2008.

Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. Henshin: advanced concepts and tools for in-place emf model transformations. In *Model Driven Engineering Languages and Systems*, pages 121–135. Springer, 2010.

Dominique Blouin, Alain Plantec, Pierre Dissaux, Frank Singhoff, and Jean-Philippe Diguet. Synchronization of models of rich languages with triple graph grammars: An experience report. In *Theory and Practice of Model Transformations*, pages 106–121. Springer, 2014.

Paul E Ceruzzi. *A history of modern computing*. MIT press, 2003.

Peter Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.

Krzysztof Czarnecki and Simon Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–645, 2006.

Zinovy Diskin. Model synchronization: Mappings, tiles, and categories. In *Generative and Transformational Techniques in Software Engineering III*, pages 92–165. Springer, 2011.

Zinovy Diskin, Arif Wider, Hamid Gholizadeh, and Krzysztof Czarnecki. Towards a rational taxonomy for increasingly symmetric model synchronization. In *Theory and Practice of Model Transformations*, pages 57–73. Springer, 2014.

Zinovy Diskin, Hamid Gholizadeh, Arif Wider, and Krzysztof Czarnecki. A three-dimensional taxonomy for bidirectional model synchronization. *Journal of Systems and Software*, 111: 298–322, 2016.

Hartmut Ehrig, Karsten Ehrig, Claudia Ermel, Frank Hermann, and Gabriele Taentzer. Information preserving bidirectional model transformations. In *Fundamental Approaches to Software Engineering*, pages 72–86. Springer, 2007.

Luciana Foss, S Costa, Nicolas Bisi, Lisane Brisolara, and Flávio Wagner. From uml to simulink: a graph grammar specification. In *14th Brazilian Symposium on Formal Methods: Short Papers*, pages 37–42, 2011.

Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, pages 37–54. IEEE Computer Society, 2007.

Holger Giese and Stefan Hildebrandt. *Efficient model synchronization of large-scale models*. Number 28. Universitätsverlag Potsdam, 2009.

Holger Giese and Robert Wagner. Incremental model synchronization with triple graph grammars. In *Model Driven Engineering Languages and Systems*, pages 543–557. Springer, 2006.

Holger Giese, Stephan Hildebrandt, and Leen Lambers. Toward bridging the gap between formal semantics and implementation of triple graph grammars. In *Model-Driven Engineering, Verification, and Validation (MoDeVVa), 2010 Workshop on*, pages 19–24. IEEE, 2010a.

Holger Giese, Stephan Hildebrandt, and Stefan Neumann. Model synchronization at work: keeping sysml and autosar models consistent. In *Graph transformations and model-driven engineering*, pages 555–579. Springer, 2010b.

Joel Greenyer, Ekkart Kindler, Jan Rieke, and Oleg Travkin. Tggs for transforming uml to csp: Contribution to the agtive 2007 graph transformation tools contest. Technical report, Department of Computer Science, University of Paderborn Paderborn, Germany, 2008.

Florian Heidenreich, Jendrik Johannes, Mirko Seifert, and Christian Wende. *Jamopp: The java model parser and printer*. Techn. Univ., Fakultät Informatik, 2009.

Florian Heidenreich, Jendrik Johannes, Mirko Seifert, and Christian Wende. Closing the gap between modelling and java. In *Software Language Engineering*, pages 374–383. Springer, 2010.

Frank Hermann, Hartmut Ehrig, Fernando Orejas, Krzysztof Czarnecki, Zinovy Diskin, and Yingfei Xiong. Correctness of model synchronization based on triple graph grammars. In *Model Driven Engineering Languages and Systems*, pages 668–682. Springer, 2011.

Stephan Hildebrandt, Leen Lambers, and Holger Giese. The mdelab tool framework for the development of correct model transformations with triple graph grammars. In *Proceedings of the First Workshop on the Analysis of Model Transformations*, pages 33–34. ACM, 2012.

Stephan Hildebrandt, Leen Lambers, Holger Giese, Jan Rieke, Joel Greenyer, Wilhelm Schäfer, Marius Lauder, Anthony Anjorin, and Andy Schürr. A survey of triple graph grammar tools. *Electronic Communications of the EASST*, 57, 2013.

Igor Ivkovic and Kostas Kontogiannis. Tracing evolution changes of software artifacts through model synchronization. In *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on*, pages 252–261. IEEE, 2004.

Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. Atl: A model transformation tool. *Science of computer programming*, 72(1):31–39, 2008.

Ivan Kurtev, Jean Bézivin, and Mehmet Akşit. Technological spaces: An initial appraisal. 2002.

Anders Mattsson, Björn Lundell, Brian Lings, and Brian Fitzgerald. Linking model-driven development and software architecture: a case study. *Software Engineering, IEEE Transactions on*, 35(1):83–93, 2009.

Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142, 2006.

Ulrich Nickel, Jörg Niere, and Albert Zündorf. The fujaba environment. In *Proceedings of the 22nd international conference on Software engineering*, pages 742–745. ACM, 2000.

Omg. Omg meta object facility (mof) core specification version 2.5. *Final Adopted Specification (June 2015)*, 2015.

OMG OMG. Unified modeling language (omg uml). *Superstructure*, 2007.

Ed Seidewitz. What models mean. *IEEE software*, (5):26–32, 2003.

Hui Song, Gang Huang, Franck Chauvel, Wei Zhang, Yanchun Sun, Weizhong Shao, and Hong Mei. Instant and incremental qvt transformation for runtime models. In *Model Driven Engineering Languages and Systems*, pages 273–288. Springer, 2011.

Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

Yingfei Xiong, Dongxi Liu, Zhenjiang Hu, Haiyan Zhao, Masato Takeichi, and Hong Mei. Towards automatic model synchronization from model transformations. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 164–173. ACM, 2007.