PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Reporte Técnico RT 12-05

Verificación de transformaciones de modelos Una Revisión del Estado del Arte

(Versión Extendida)

Daniel Calegari   Nora Szasz

2012

Verificación de transformaciones de modelos. Una Revisión del Estado del Arte (Versión Extendida)
Calegari, Daniel; Szasz, Nora
ISSN 0797-6410
Reporte Técnico RT 12-05
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay, 2012

Verificación de Transformaciones de Modelos

Una Revisión del Estado del Arte

(Versión Extendida)

Daniel Calegari[1] and Nora Szasz[2]

[1] Facultad de Ingeniería, Universidad de la República, 11300 Montevideo, Uruguay
dcalegar@fing.edu.uy
[2] Facultad de Ingeniería, Universidad ORT Uruguay, 11100 Montevideo, Uruguay
szasz@ort.edu.uy

Abstract. Dentro del paradigma de Ingeniería Dirigida por Modelos el desarrollo de software se basa en la definición de modelos que proveen diferentes vistas del sistema a construir y transformaciones de dichos modelos que soportan un proceso de desarrollo (semi)automático. La verificación de los modelos y de las transformaciones es crucial a los efectos de mejorar la calidad y confiabilidad de los productos desarrollados utilizando este paradigma. En este contexto, la verificación de una transformación de modelos tiene tres componentes: la transformación en sí misma, las propiedades de interés involucradas y las técnicas de verificación utilizadas para establecer las propiedades. En este artículo presentamos una revisión exhaustiva de la literatura existente sobre verificación de transformaciones de modelos, analizando estos tres componentes. Además, tomamos un enfoque basado en problemas, ejemplificando los aspectos de interés que podrían ser verificados en una transformación de modelos y cómo estos podrían serlo. Finalmente concluimos sobre la necesidad de contar con un ambiente integrado para llevar a cabo la verificación heterogénea de transformaciones de modelos.

Palabras clave: ingeniería dirigida por modelos, transformaciones de modelos, verificación formal

Verification of Model Transformations[*]

A Survey of the State-of-the-Art
(Extended Version)

Daniel Calegari[1] and Nora Szasz[2]

[1] Facultad de Ingeniería, Universidad de la República, 11300 Montevideo, Uruguay
`dcalegar@fing.edu.uy`
[2] Facultad de Ingeniería, Universidad ORT Uruguay, 11100 Montevideo, Uruguay
`szasz@ort.edu.uy`

Abstract. Within the Model-Driven Engineering paradigm, software development is based on the definition of models providing different views of the system to be constructed and model transformations supporting a (semi)automatic development process. The verification of models and model transformations is crucial in order to improve the quality and the reliability of the products developed using this paradigm. In this context, the verification of a model transformation has three main components: the transformation itself, the properties of interest addressed, and the verification techniques used to establish the properties. In this paper we present an exhaustive review of the literature on the verification of model transformations analyzing these three components. We also take a problem-based approach illustrating those aspects of interest that could be verified on a model transformation and show how this can be done. Finally, we conclude the need of an integrated environment for addressing the heterogeneous verification of model transformations.

Keywords: model-driven engineering, model transformations, formal verification

[*] Technical Report / InCo-PEDECIBA / TR12-XX / ISSN 0797-6410 / June 2012 http://www.fing.edu.uy/inco/pedeciba/bibpm/field.php/Main/ReportesT%e9cnicos

1 Introduction

Every traditional software development life-cycle is supported by a number of artifacts (e.g. requirements specifications, analysis and design documents, test suites, source code) which are mostly used as guides for the development as well as communication tools with the stakeholders.

The Model-Driven Engineering (MDE) [Ken02,Sch06] paradigm pushes this view to its limits by envisioning a software development life-cycle driven by artifacts which are models representing different views of the system to be constructed. Its feasibility is based on the existence of a (semi)automatic construction process driven by model transformations, starting from abstract models of the system and transforming them until an executable model is generated. In consequence, the quality of the whole process strongly depends on the quality of the models and model transformations.

We are concerned with model transformations and particularly with their verification. In this sense, the minimal requirement to be verified on a model transformation is that the transformation and the source and target models are well-formed. However, there are multiple other properties that could be verified and there is a plethora of verification approaches to do so. This topic is analyzed in [ALS$^+$12] as a tri-dimensional problem consisting of: the transformation involved, the properties of interest addressed, and the formal verification techniques used to establish the properties.

The aim of this paper is to present a comprehensive review of the literature on the verification of model transformations extending the work in [ALS$^+$12]. Particularly, we introduce the first dimension without going deeper, since there are well-known works [MCG04,CH06,Men10] addressing this subject, and we extend the second and third dimensions with other aspects not addressed in [ALS$^+$12]. We also follow a problem-based approach exemplifying by a case study those aspects of interest that could be verified on a model transformation and how they can be verified. Finally, we conclude the need of an integrated environment for addressing the heterogeneous verification of model transformations.

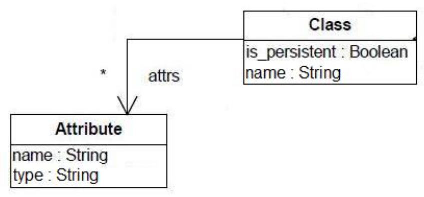
The remainder of the paper is structured as follows. We first take a quick look at model transformations in Section 2. Then, in Section 3 we introduce the different aspects of a transformation that must be verified, and in Section 4 we review how these aspects are verified in the literature. In Section 5 we define some cases to exemplify verification properties and discuss how they can be verified. Finally, in Section 6 we present some thoughts on this topic and guidelines for future work. Additionally, in Appendix A we include a comprehensive annotated bibliography on verification of model transformations.

2 A Quick Look at Model Transformations

In the MDE ecosystem everything is a model, even the code is considered as a model. In this context, a model is an abstraction of the system or its environment. Every model *conforms* to a metamodel, i.e. a model which introduces the syntax and semantics of certain kind of models. In the same way, a metamodel conforms to some metameta-model. A metametamodel is usually self-defined, which means that it can be specified by means of its own semantics.

Metamodels are usually defined using Unified Modeling Language (UML) Class Diagrams [OMG05]. However, there are several other specific languages for this purpose, e.g. the MetaObject Facility (MOF) [OMG03], the Ecore metametamodel defined for the Eclipse Modeling Framework (EMF) [FSM$^+$03], and Kernel MetaMetaModel (KM3) [ATL05] defined for the Atlas Transformation Language (ATL) [JK05]. Besides a metamodel defines a modeling language which usually has a concrete syntax, it is possible to represent a model using the same languages as for metamodels. Moreover, for representing model instances, as well as for models which are a kind of "instance" of a metamodel, there is the graphical representation provided by UML object diagrams. Finally, every element in the hierarchy could be represented using XML Metadata Interchange (XMI) [OMG11]. In some cases, there are conditions (called invariants) that cannot be captured by the structural rules of these languages, in which case modeling languages are supplemented with another logical language, e.g. the Object Constraint Language (OCL) [OMG10].

As an example, consider the metamodel in Figure 1 defining a simple UML Class Diagrams such that classes can contain one or more attributes (with a name and a type) and may be declared as persistent. The metamodel is defined as a UML Class Diagram and also using KM3.



```
class Class{
  attribute name : String;
  attribute is_persistent : Boolean;
  reference attrs[*] container : Attribute;
}

class Attribute{
  attribute name : String;
  attribute type : String;
}
```

Fig. 1. Simple class metamodel

The second building block of the MDE paradigm is the model transformation, which could also be considered as a model. As pointed out in [CH06], model transformation is closely related to program transformation. "Their differences occur in the mindsets and traditions of their respective transformation communities, the subjects being transformed, and the sets of requirements being considered. While program transformation systems are typically based on mathematically oriented concepts such as term rewriting, attribute grammars, and functional programming, model transformation systems usually adopt an object-oriented approach for representing and manipulating their subject models."

Kleppe et al. [KWB03] define a model transformation as follows: "A *transformation* is the automatic generation of a target model from a source model, according to a transformation definition. A *transformation definition* is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. A *transformation rule* is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language."

As summarized in Figure 2, extracted from [ATL06], a model transformation basically takes as input a model *Ma* conforming to a given source metamodel *MMa* and produces as output another model *Mb* conforming to a given target metamodel *MMb*. The model transformation can be defined as well as a model *Mt* which itself conforms to a model transformation metamodel *MMt*. This last metamodel, along with the *MMa* and *MMb* metamodels, must conform to a metametamodel (such as MOF or Ecore). The transformation definition is executed by a transformation engine.

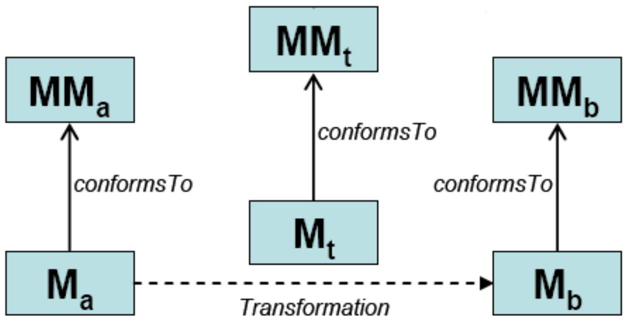


Fig. 2. An overview of model transformation

This schema defines model-to-model transformations. There are also model-to-text and text-to-model transformations where the target and source models, respectively, are just strings not conforming to any specific metamodel. Without loss of generality we will only consider model-to-model transformations (from now just transformations, or model transformations).

As pointed out in [MCG04,Men10], "this definition is very general, and covers a wide range of activities for which model transformation can be used: automatic code generation, model synthesis, model evolution, model simulation, model execution, model quality improvement (e.g. through model refactoring), model translation, model-based testing, model checking, model verification."

However, this schema can be extended as is exhaustively studied in [CH06]. Leaving aside the details, the authors identify multiple variabilities on a model transformation, e.g. it can be bidirectional, it can take more than one source model as input and/or produce multiple target models as output, it rule application strategy can be deterministic, non-deterministic or interactive, the source and target models could be or not at different abstraction levels (horizontal versus vertical transformations), and the source and target models could conform or not to the same metamodel (endogenous versus exogenous transformations).

Beyond these aspects, there are several approaches for defining and executing model transformations:

Direct-manipulation. It offers an internal model representation and some Application Programming Interface (API) to manipulate it. In this case the transformation must be usually developed in a programming language, like Java. An example of this is the Java Metadata Interface (JMI) [Dir02].

Structure-driven. It consists of two distinct phases: first a hierarchical structure of the target model is created from the transformation rules, then the attributes and references are set in the target. An example of this is the framework provided by OptimalJ [Com05].

Operational (a.k.a. Imperative). It is similar to direct manipulation but offers more dedicated support. A typical solution is to extend the used metamodeling formalism with facilities for expressing computations. Examples of systems in this category are Query/View/Transformation (QVT) Operational mappings [OMG09], and Kermeta [MFJ05].

Template-based. Model templates are models with embedded metacode that computes the variable parts of the resulting template instances. The metacode can have the form of annotations on model elements. An example of this approach is the one introduced in [CA05].

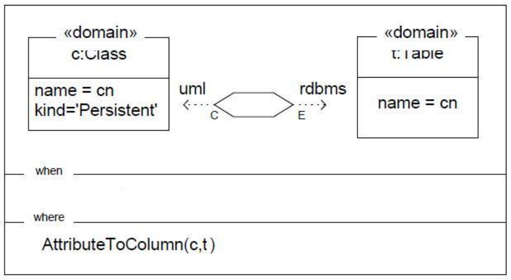
Relational (a.k.a. Declarative). It consists of defining transformation rules as mathematical relations between source and target elements. Its execution can be seen as a form of constraint solving. Examples of this approach are QVT Relations, and Tefkat [LS05].

Graph-transformation-based. It consists of considering models as typed, attributed, labeled graphs and applying graph transformations. Examples of this include the following tools: Attributed Graph Grammar (AGG) [Tae03], VIsual Automated model TRAnsformations (VIATRA) [CHM$^+$02], and Graph Rewriting and Transformation (GReAT) [AKL03].

Hybrid approaches. It concerns the combination of different techniques from the previous approaches. An example of this are QVT and ATL which combine operational and relational strategies.

Others approaches. Two more approaches are mentioned for completeness: transformation implemented using Extensible Stylesheet Language Transformation (XSLT) [Kay05] and the application of metaprogramming.

As an example, consider the transformation in Figure 3 which represents the transformation from classes (specified using the metamodel in Figure 1) to relational tables, as well as attributes to columns. The transformation rules are represented in QVT Relational graphical notation (left) and in ATL declarative notation (right).



```
rule ClassToTable{
  from c : SimpleUML!Class(c.is_persistent)
  to t : SimpleRDBMS!Table (
           name <- c.name,
           cols <- c.attrs
         )
}

rule AttributeToColumn{
  from a : SimpleUML!Attribute ()
  to c : SimpleRDBMS!Column (
           name <- a.name,
           type <- a.type
         )
}
```

Fig. 3. Model transformation from Class to Table

A deeper comparison between transformation languages and tools can be found in [LVV+09].

3 What to verify?

As we have seen in the last section, there are many transformation approaches. However, these approaches do not define what things have to be verified but how in some cases the verification must be done. Consequently, in this section we will explore transformation properties without talking about any transformation approach in particular.

In this section we will focus on the second dimension introduced in [ALS+12]: the properties of interest addressed by the verification of a model transformation. There are also other works [KAER06,VP03] which introduce the problem of verification by defining the set of properties to be addressed. However, the contents of these proposals are mostly included in the former one. We thus present the categories of properties identified in [ALS+12]: language-related and transformation-related properties. Next we explain each category without deeping inside those aspects already presented in [ALS+12], and extend them when necessary for adding more (sub)categories. We also include properties addressed in other works with a standardized nomenclature.

3.1 Language-Related Properties

This category refers to the computational nature of transformations and target properties of transformation languages. As introduced in [ALS+12], a transformation specification conforms to a transformation language which can possess properties on its own. In this context there are four properties of interest.

The first two properties are identified as execution-time properties. They are the **Termination** property which guarantees the existence of a target model, i.e. that the transformation execution finishes for any well-formed transformation specification, and the **Determinism** (a.k.a. Confluence) property which ensures uniqueness of the target model for a given source model and transformation specification. These properties are related to undecidable problems for sufficiently expressive (i.e. Turing-complete) transformation languages. In these cases, "[...] formally proving them cannot be done by relying on one particular transformation's specifics. [...] either the TL [Transformation Language] is kept as general (and powerful) as possible, making these properties undecidable, but the transformation framework provides capabilities for checking sufficient conditions ensuring them to hold on a particular transformation; or these properties are ensured by construction by the TL, generally by sacrificing its expressive power" [ALS+12]. According to this, in the first case, the properties could also be identified as a transformation-related properties (as defined in the next section) when proved for a specific transformation specification.

The third property, identified as a design-time property, is **Typing**, i.e. ensuring the well-formedness of the transformation specification w.r.t. its transformation language. The process of type checking may occur either at compile or run-time. Since model transformations are models, and models have metamodels (defining the transformation language), solutions to this problem are strongly related to Conformance and Model Typing as will be introduced in the next section.

Finally, we introduce a fourth property, not mentioned in [ALS+12], the **Preservation of Execution Semantics** property. This execution-time property states that the transformation execution must behave as expected according to the definition of the transformation language semantics. Related to this, and in strong contact with the Typing property, there are consistency needs between transformation rules which must also hold. For example, some languages do not allow an element of the input model to be matched more than once (redundancy problem). If this property does not hold, contradictory rules may be applied, e.g. two rules applied to the same element implying different things. Moreover, it is possible that a rule applied to an element of a hierarchy may be more restrictive than other one applied to an element in a lower level of the same hierarchy. In this case, it will be some models not matched by the second rule.

3.2 Transformation-Related Properties

This category refers to the modeling nature of transformations. As introduced in [ALS+12], a transformation refers to source/target models for which dedicated properties need to be ensured for the transformation to behave correctly.

In this context there is a first step on verification that strictly concerns the source and/or target model(s) a transformation refers to. This subcategory of properties is known as **On the Source/Target Model(s)**. As pointed out in [VP03] "the minimal requirement is to assure syntactic correctness, i.e., to guarantee that the generated model is a syntactically well-formed instance of the target language [w.r.t. structural and non-structural constraints]". This introduces a first group of properties known as **Conformance and Model Typing**. Conformance is nowadays well understood and automatically checked within modeling frameworks. There is a second group known as **N-Ary Transformations Properties**: transformations operating on several models at the same time, e.g. model composition, merging, or weaving, require dedicated properties to be checked.

But verification interests go beyond this kind of problems. When verifying a model transformation we want to consider its elements as a whole and not individually. In this sense, some authors, as in [CCGdL09], use the notion of a transformation model, i.e. a model composed by the source and target metamodel, the transformation specification and the well-formedness rules. This transformation model could be implicit, i.e. we assume that every element is connected, or explicit, i.e. we really construct a unified structure with different purposes (e.g. tracing or verification). This model states how elements are related, and these relations introduce some syntactic and semantic questions.

In this sense, there are properties known as **Model Syntax Relations** that relate metamodel elements of the source and the target metamodels trying to ensure that certain elements or structures of any input model will be transformed into other elements or structures of the output model. This problem arises when these relations cannot be inferred by just looking at the individual transformation rules, or when the transformation language does not allow expressing some relations, and another constraint language must be used. This is also known as preservation of transformation invariants or structural correspondence.

Beyond structural relationships between source and target models, there are semantic properties that must be preserved, known as **Model Semantics Relations** and also as *semantic correctness* or *dynamic consistency* [VP03]. These properties generally depend on the metamodels semantics or on the kind of transformation. Some properties of interest are semantic equivalence, (weak) bisimilarity and preservation of properties, temporal properties, refactoring, and refinement.

Finally, we add a fourth category called **Functional Behavior**, not mentioned in [ALS+12], which refers to identifying if a transformation behaves as a mathematical function. In particular, it is possible that a transformation may be injective, surjective, bijective, or at least, executable (i.e. there exists a valid pair of source and target models that satisfy the transformation). It is also possible to analyze these properties considering individual rules within a transformation. These properties are introduced in [CCGdL09]. Moreover, there is a specific property known as **Syntactic Completeness** which refers to the need (in some cases) of completely covering the source/target metamodel by transformation rules. This is also presented in [KAER06] as *metamodel coverage* introducing the problem that if the transformation does not cover the entire metamodel, then this leads to some input models which cannot be transformed. From a functional point of view, syntactic completeness means that the transformation is a total function. When considered for a specific transformation, determinism is also a functional property. In fact, as introduced in [CCGdL09], when a transformation is *total* and *deterministic*, it is called *functional*.

3.3 Concluding Remarks

We have seen a classification of the properties of interest addressed by the verification of a model transformation. This classification, formerly introduced in [ALS+12], identifies language-related and transformation-related properties, the first ones referring to the computational nature of transformations and target properties of transformation languages, and the second ones referring to the modeling nature of transformations. We extended this classification by adding two subcategories addressing properties based on other related works.

When following a MDE-based software project, formal verification is mostly focused on the second category of properties (transformation-related), whilst those within the first category (language-related) are in general assumed to be somehow automatically verified by the development tools.

A summary of the properties addressed in the literature can be found in Table 1.

Table 1. Summary of properties addressed in the literature

Language-Related Properties		
Termination	[BLA+10][Bru08][EEdL+05][Küs06][LR10][VVGE+06][WKK+09]	
Determinism	[BLA+10][CCGdL09][HEOG10][HKT02] [Küs06][LEO06][LR10][WKK+09]	
Typing	[KMS+09][LLM09],[SJ07]	
Preservation of Exec. Sem.	[ABGR10][CCGdL09][GdLW+12] [LD10][PCG11][WKK+09]	
Transformation-Related Properties		
Source/Target	Conformance	[ABGR10][AKP03][ALL10] [CBBD09][CLST10][GM07][Küs04] [LD10][LMAL10][LR10][LR11][Poe08] [Sch10][SMR11][WKK+09]
	N-Ary	[CNS12][Kat06][MSJ+10]
Syntax Relations	[AKP03][ALL10][CBBD09][CLST10][GM07] [CHM+02][GdLW+12][LBA10][LD10][LMAL10][LR10] [NK08a][OW09][Poe08][Sch10][SK08]	
Semantics Relations	General	[CLST10][LR11][Poe08][Sch10][SMR11]
	Sem. Eq.	[BEH06][BKMW08][CCGT09][GGL+06][HHK10][LR10] [NK08b][PGE97][RLK+08][VP03]
	Temporal	[BBG+06][BHM09][CHM+02][EKHL03]
	Refactoring	[HT05]
	Refinement	[CBBD09][MGB05][PG08]
Functional Behavior	General	[CCGdL09]
	Synt. Comp.	[CHM+02][GdLW+12][KAER06][LR10][PCG11][WKC06]

4 How to verify?

As pointed out in [EKHL03], a property can be either verified or validated, leading to the well-known distinction between verification and validation. Formally, verification is addressed to "determine whether the products [...] satisfy the conditions imposed" whilst validation is addressed to "determine whether it [the product] satisfies specified requirements" [IEE90]. In other words, verification is the process of proving that we are building the product in the right way, while validation is the process of proving that we are building the right product.

We are focused on verification, and in particular on formal verification, i.e. in the act of verifying using formal methods. Although formal verification techniques may be expensive, they can be helpful in guaranteeing the correctness of critical applications where no other verification technique is acceptable. In contrast to formal verification, there are other techniques which may detect errors or improve confidence, but they cannot prove any property in a definite way.

In this section we will focus on the third dimension introduced in [ALS$^+$12]: the formal verification techniques used to establish the properties. A very initial version of this review can be found at [LVV$^+$10].

4.1 Inference, model checking, testing, static analysis or by construction

We introduce this first category which refers to the kind of technique used for verification. **Logical inference** (a.k.a. **theorem proving**) consists of using a mathematical representation of a system and the properties that must be verified, as well as a logic in that semantic domain which allows reasoning about that representation, leading from premises to conclusions. This process is usually carried out using theorem proving software and it is usually only partially automated. **Model checking** also consists of using a mathematical representation of a system and proofs consist of a systematic exhaustive exploration of the mathematical model. With the first approach there is usually a high verification cost, whilst with the second there are well-known limitations such as the state-explosion problem. On the other hand, **testing** relies on the construction of test strategies for a property including subsequent execution of (either parts or all of) the system according to these strategies. Although testing is usually considered a validation strategy, it could be used for verification purposes. However, as it is popularly said, testing can only show the presence of errors and not their absence. Finally, we can find strategies based on **static analysis**, i.e. on the analysis of a model transformation that is performed without actually executing it. Static analysis typically consists on semi-decision techniques. In this sense, they are efficient but they cannot assure the overall correctness of the design. For matter of completeness we also consider the satisfaction of properties that hold by construction of the transformation, e.g. those achieved by using special transformation languages as DSLTrans [BLA$^+$10].

4.2 Metamodel or model level

This category consists of the abstraction level w.r.t. the elements involved in the transformation, and it is also referred as *offline and online* verification [ALL10], and as *input independent and input dependent* verification [ALS$^+$12]. **Metamodel-level** verification uses the metamodel information for verifying properties for any well-formed model instance while **model-level** verification uses arbitrary source models. As pointed out in [VP03], the first level typically requires the use of sophisticated theorem proving techniques and tools with a huge verification cost. For this reason, the second is in many cases a practical and valuable aid, but it cannot ensure the zero-fault level of quality since it checks a finite number of specific cases. However, as model-level verification takes place on a lower level of abstraction, the range of properties that can be validated is much greater than when using metamodel-level verification.

4.3 Specification or implementation

As introduced in [EKHL03], verification can either be done on the model (specification) level or on the implementation level. **Specification-level** verification involves only the specification of the transformation in some transformation language, and in consequence the semantics defined for that transformation language. In contrast, **implementation-level** verification means also considering the way a transformation is executed by a transformation engine. As far as we know, verification techniques found in the literature are of the first type, since it is assumed that any transformation engine conforms with the semantics of the transformation language and properties do not depend on how exactly the transformation is executed, including the case of Determinism, Termination and Preservation of Execution Semantics properties.

4.4 Transformation independent or dependent

This dimension is introduced in [ALS$^+$12]. **Transformation independent** techniques are those techniques which prove properties for any transformation, and in consequence they assure that no assumption is made on the specific source model. In contrast, **transformation dependent** techniques rely on a specific model transformation. Transformation independency is achieved either by a transformation language that preserves the properties by default, or by ensuring a property by construction of the transformation.

4.5 Concluding Remarks

We have shown how verification techniques can be classified in different categories referring to: (a) the kind of technique used for verification, (b) the abstraction level w.r.t. the elements involved in the transformation, (c) the abstraction level w.r.t. the implementation of the transformation, and (d) the dependency/independency w.r.t the transformation specification. It is worth saying that these categories are orthogonal, i.e. there are verification techniques which correspond to more than one category. A summary of the verification techniques addressed in the literature within this categorization can be found in Table 2.

Table 2. Summary of verification techniques addressed in the literature

	inf	mod chk	sta ana	test	cons	meta	mode	spec	impl	tran ind	tran dep
[ABGR10,BBG$^+$06]	√						√	√			√
[ALL10,LMAL10,GGL$^+$06]	√					√		√			√
[Bru08,Cha06,CLST10,LR10]	√					√		√		√	
[LD10,Poe08,Sch10,SMR11]	√					√		√		√	
[AKP03,CNS12]	√						√	√		√	
[PG08]	√	√		√			√	√			√
[CCGdL09,LEO06,EEdL$^+$05]		√				√		√		√	
[MSJ$^+$10,HEOG10,HKT02]		√				√		√		√	
[OW09,VVGE$^+$06,WKK$^+$09]		√				√		√		√	
[LBA10,VP03]		√				√		√			√
[BEH06,BHM09,CBBD09]		√					√	√		√	
[CHM$^+$02,EKHL03]		√					√	√		√	
[GdLW$^+$12,GM07]		√					√	√		√	
[HHK10]		√						√		√	
[HT05,MGB05,RLK$^+$08]		√				√	√	√		√	√
[Küs04,Küs06]		√	√			√		√		√	
[NK08b,NK08a]		√				√	√	√			√
[Kat06,VR11]			√					√		√	
[PCG11]			√			√		√		√	
[WKC06,KAER06]				√			√	√		√	
[BKMW08,BLA$^+$10,KMS$^+$09]					√	√		√		√	
[LLM09,PGE97,SJ07]					√	√		√		√	
[CCGT09]					√	√		√			√

5 Verification by Example

In this section we introduce a couple of examples to illustrate several verification properties and discuss how the verification could be addressed. These examples were extracted from the literature and adapted to keep certain homogeneity in the definitions.

5.1 Class to Relational

The first example is the well-known Class to Relational model transformation, originally introduced in [BRST05], which became the de-facto standard example for model transformations. There are other versions, as for example the one in the QVT specification. We will use the transformation as presented in [JK05].

The metamodel in Figure 4 defines UML class diagrams, where classes can contain one or more attributes, can belong to a class hierarchy and may be declared as persistent (attribute is_persistent). Each attribute has a type that can be another class or a primitive datatype (string, boolean, integer, etc.). Attributes may be defined as primary (attribute is_primary). Associations are defined between classes with a direction from source to destination. An additional constraint is imposed that every class must have at least one attribute and at least one primary attribute (they may be inherited).

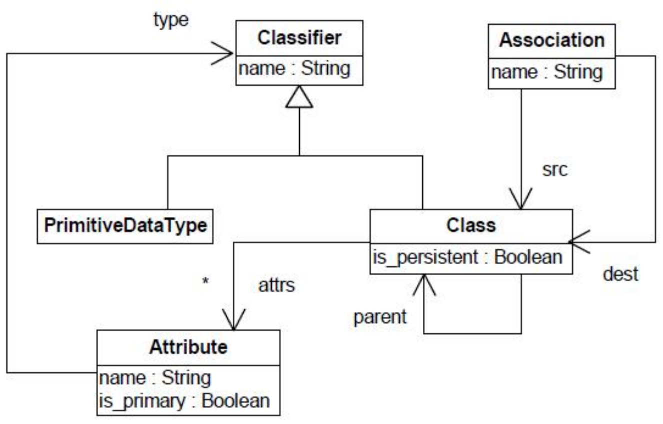


Fig. 4. Class metamodel

On the other side, relational models conform to the metamodel in Figure 5. Every model contains a number of tables and each table has a number of columns. Some of these columns are primary keys of the corresponding table. A table may be associated to zero or more foreign keys. Each foreign key refers to a table and is associated with a number of columns that constitute the key.

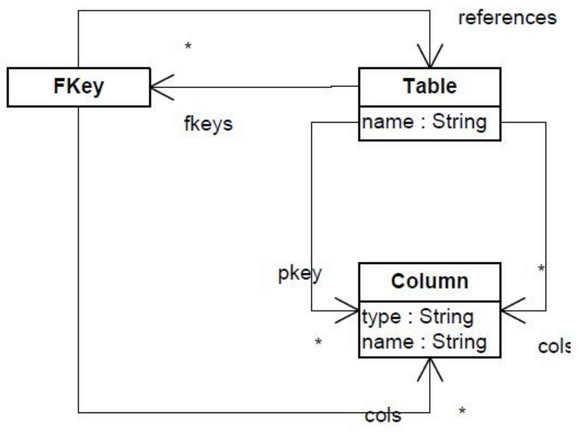
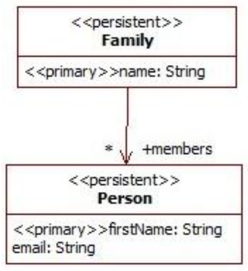


Fig. 5. Relational metamodel

In our running example, the transformation basically describes how persistent classes are transformed into tables. Attributes and associations of a class are transformed into columns of the corresponding table, and the primary and foreign keys are also set appropriately. Primary keys are defined for attributes defined as primary, as well as foreign keys for associations with other classes, including those in the class hierarchy. Below we show the persistent class to table rule definition using ATL declarative notation. The transformation uses a pre-processing step which flattens the features (either attributes or outgoing associations) of classes in a hierarchy. With this intermediate structure it is easier to define the rules, as the following rule which maps persistent classes to tables.

```
rule PersistentClass2Table{
  from
    c : SimpleClass!Class (
      c.is_persistent and c.parent.oclIsUndefined()
      )
  to
    t : SimpleRDBMS!Table (
      name <- c.name,
      cols <- c.flattenedFeatures->select(f |
        not f.isForeignKey
        )->collect(ft | ft.trace),
      pkey <- c.flattenedFeatures->select(f |
        f.isPrimary)->collect(ft | ft.trace),
      fkeys <- c.flattenedFeatures->select(f |
        f.isForeignKey)
        )
}
```

In Figure 6 there is an example of a source model (in UML class diagram notation with stereotypes) and its corresponding target model (in KM3-like notation).



```
table Family {
  primary column name : String;
}

table Person {
  primary column firstName : String;
  column email : String;
}

table Family_members {
  column familyName : String;
  column personFirstName : String;
}
```

Fig. 6. Source and target models for the Class to Relational transformation

Let us now define which kind of properties could be stated within this example.

Conformance and Model Typing Beyond the basic conformance needs, there are usually invariants that cannot be captured by the structural rules of the modeling language. Invariants are well-formedness rules that must hold at all time for any model conforming to a metamodel. In our case, we have invariants on metametamodels, metamodels and models. In the example the following invariants must hold:

- All associations have distinct names (the same for classes)
- The owned attributes of a class are uniquely named within their owner class and the classes it inherits
- There are no cycles of inheritance within the parent relation in classes
- If a class is persistent so are all of its superclasses
- If a class is persistent it must have at least one attribute marked as primary
- All tables have distinct names
- All columns have distinct names within a table
- Every table must have at least one primary column

Moreover, possible invariants on models are for example:

- All families have distinct names
- Every person has a distinct first name within a family

Invariants could be expressed using a constraint language like the OCL, and as it was said in Section 3, this conformance checking is nowadays automatically addressed within modeling frameworks using automated checkers. These checkers can be based on SAT solvers or model-checking, as in [ABGR10,GM07]. This verification is at a model level. But there are other alternatives, for example performing the verification using logical inference. In this case, we can formalize metamodels, models and invariants in some formal language, like B or directly First-Order Logic (FOL), and then use a proof assistant, like Coq and Isabelle/HOL, as done in [LD10,LR10,Sch10,SMR11].

Moreover, using this strategy, one could perform the verification at a metamodel level, forgetting models and considering transformations and proving the postconditions assuming that both the pre-conditions and the transformation rules hold, as done in [CLST10]. For our running example, a simple property that can be proven is that the length of the `Columns` within a `Table` must be greater than zero. This property holds by the fact that every `Attribute` is transformed into a `Column` and because every `Class` has at least one `Attribute`. This information is given in the transformation rules and in the source invariants, respectively.

A step further of this approach is the one presented in [Poe08] where giving a representation of models, metamodels and transformations in the Calculus of Inductive Constructions [CP90], they can extract a correct transformation. This requires specifying a transformation as types of the form

$$\forall x \,:\, Pil.I(x) \,\rightarrow\, (\exists y \,:\, Psl.O(x, y))$$

where Pil and Psl are source and target metamodel types, $I(x)$ specifies a precondition on the source model x for the transformation to be applied, and $O(x, y)$ specifies required properties of the output model y. A proof of this expression implies the automatic construction of a function f such that, given any x satisfying the precondition $I(x)$, then the postcondition $O(x, fx)$ will be satisfied.

Finally, a complementary approach in [LMAL10] proposes a language for assertions based on FOL that describes some characteristics of a model under transformation. Then, they can derive how an assertion evolves when applying transformation rules using SWI-Prolog [WSTL10] as an inference system. If the assertions to be verified can be derived from the final assertion, thus they hold in the target model.

Model Syntax Relations A transformation model gives useful information about the relation between the elements connected by a transformation. With this, some other relations could be inferred which are not evident by just looking at the individual elements.

In our example we can illustrate this with the following property: if `c` is a subclass of `d`, all columns of `c`'s table are included in `d`'s table. Since the rule `PersistentClass2Table` uses the flattening of features of the source metamodel, this property cannot be trivially inferred.

There are many alternatives for proving this property. First, we can use a formal language to state it and a proof assistant to prove it. We can also use the language for assertions and prove that this property can be derived from the final assertion. We explored these alternatives when discussed conformance and model typing.

Another option is to define a transformation contract stating the pre and post conditions of a transformation (or an individual rule), and check whether this contract holds. This contract could be written in OCL and then verified using an OCL checker or some other model-checker, as in [CBBD09,GM07]. It can also be written using a dedicated tool and then verified using some specific algorithm, as in [CHM$^+$02,GdLW$^+$12].

Functional Behavior As defined in [CCGdL09], it is possible to check whether some properties hold considering either a specific rule or the whole model transformation. As an example, the rule `PersistentClass2Table` is executable since there exists a valid pair of models (those in Figure 6) that satisfy it. Since this rule is the top rule in the transformation, we can derive that the whole transformation is also executable.

Moreover, the same rule is not injective. Consider that the transformation maps persistent classes that are roots of a hierarchy to tables, whilst derived classes are flattened and their attributes mapped to columns of the former table. Then, we can find a counterexample which has the same target (a table with two columns) where one is produced from a class with two attributes, and the other from two classes related through inheritance with one attribute each, as shown in Figure 7.

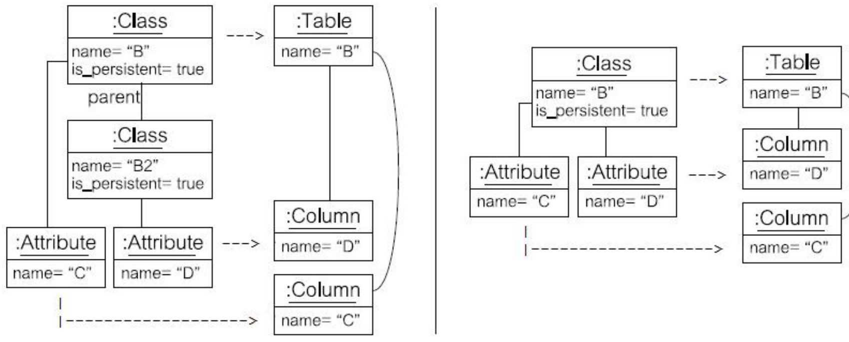


Fig. 7. Counterexample of injectivity

Following the same ideas, neither the rule nor the transformation is total (Syntactic Completeness) since they apply only for persistent classes and then non-persistent classes will never be transformed. In this case it is clear that syntactic completeness is not desirable.

These properties could be verified by encoding them as UML/OCL consistency problems on the transformation model, as defined in [CCGdL09]. Then, an OCL-checker could be used. Another alternative is to verify them by static analysis of the transformation rules and the underlying metamodels, as in [PCG11].

5.2 Determinism and Termination

As we already said in the previous section, these two properties are the hardest to prove since there are related to undecidable problems. One alternative to achieve them is to use some language which guarantees both by construction, as introduced in [BLA$^+$10]. However, this option clearly reduced expressive power.

Other alternatives are representing the transformation model in some formal language to perform logical inference, as in [LR10], or using static analysis, as in [Küs06].

However, the most referred alternative is to express the transformation as a graph-rewriting problem which allows performing critical pair analysis, as in [Bru08,EEdL+05,HEOG10,HKT02,LEO06,VVGE+06].

Preservation of Execution Semantics The preservation of the execution semantics is matter of verification during the development of a transformation engine. However, when defining a model transformation there are consistency needs that must be addressed. As an example, we may not want redundant rules, and indeed our example does not have redundancy. An example of a redundant rule would be one mapping attributes to columns but applicable only to persistent classes.

As before, we can encode these needs as a consistency problem on the transformation model and use a model-checker, as in [ABGR10], or use static analysis, as in [PCG11]. Moreover we can use logical inference as in [LD10].

Another possible approach is the followed in [WKK+09], where the transformation is expressed as a Colored Petri Net (CPN) which allows the formal exploration of CPN properties. In particular, the authors can verify whether there are transitions which are never enabled during execution, so called Dead Transition Instances or L0-Liveness.

5.3 UML Statecharts to Petri Nets

With the next example we will show how behavioral models introduce other interesting properties to verify. This model transformation from UML Statecharts into Petri Nets, formerly introduced in [VP03], is aimed at formal verification of safety critical applications defined using UML statecharts by semi-decision analysis methods of Petri nets.

A simplified version of a statechart metamodel is defined in Figure 8. Statecharts basically consist of states and transitions between them. A state may be simple or complex, i.e. containing other states that may execute in parallel. A transition connects a source state to a target state. It includes (optionally) a guard, a sequence of output actions, and it is triggered by an event. There are also some special kind of states called pseudostates, e.g. join and fork pseudostates are used to collect different transitions into a compound transition.

The metamodel in Figure 9, represents a Petri Net. Petri nets are bipartite graphs, with two disjoint sets of nodes: `Places` and `Transitions`. Places may contain an arbitrary number of tokens. The state of the net can be changed by firing enabled transitions. A transition is enabled if each of its input places contains at least one token (if no arc weights are considered). When firing a transition, we remove a token from all input places (connected to the transition by `InArcs`) and add a token to all output places (as defined by `OutArcs`).

The model transformation was designed for projecting UML statecharts into Petri Nets in order to carry out various kinds of formal analysis. As explained in [VP03], in this transformation each statechart state is modeled with a respective place in the target

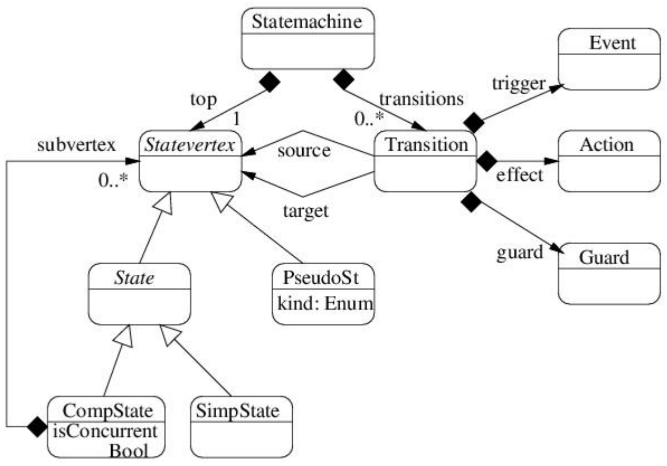


Fig. 8. Statechart metamodel

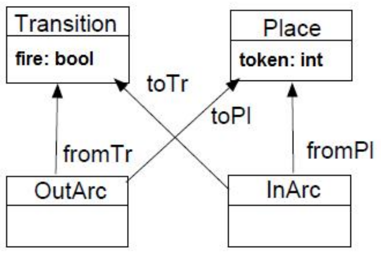


Fig. 9. Petri net metamodel

Petri Net model. A token in such a place denotes that the corresponding state is active, therefore, a single token is allowed on each level of the state hierarchy. In addition, places are generated to model messages stored in event queues of a statemachine. Each statechart step is projected into a Petri Net transition. When such a transition is fired, (i) tokens are removed from source places and event queue places, and (ii) new tokens are generated for all the target places and receiver message queues. Therefore, input and output arcs of the transition should be generated in correspondence with this rule.

As may be expected, this transformation also has verification needs as those explained in the last example, e.g. invariants in the source and target metamodels, model syntax relations that may be ensured, and so on. Since these properties and the strategies used for verifying them were already exemplified in the Class to Relational example, we omit them here and concentrate on model semantic relations.

Model Semantic Relations Since we are working with behavioral models, it is easy to find semantic properties that must be preserved. In this example we have a safety criterion for statecharts (temporal logic property) which can be stated as: "For all non-concurrent composite states in a UML statechart, only a single substate is allowed to be active at any time during execution". Since we have defined a semantic-preserving transformation, this property must also hold in the target model.

There are several techniques to prove this, which can be categorized as inference-based strategies [CLST10,Lan06,Poe08,Sch10], and model-checking strategies [BHM09,VP03]. The first category contains the same strategies used for verifying model syntax relations. However, these strategies have some problems when dynamic properties are involved, since they must not only represent the structure but also how this structure behaves. In contrast, model checking is intended to be used for proving dynamic properties at a model level.

Any two source and target models may be translated into behaviorally equivalent transition systems, as in [VP03]. Then the property could be defined in the source transition system and then transformed into a semantically equivalent property in the target transition system (manually, or using the same transformation program). Finally both transition systems are model-checked automatically to prove the properties. If the verification succeeds, then it is possible to conclude that the model transformation is correct with respect to the pair of properties for the specific pairs of source and target models.

A similar approach is followed in [BHM09] where transformation elements are formalized in rewriting logic and thus Maude's reachability analysis and model checking features can be used for verifying temporal properties.

6 Concluding Remarks

We conducted a review of the literature on the verification of model transformations. This review was structured following the three dimensions presented in [ALS$^+$12]. Furthermore, we conducted a comprehensive literature review extending this work and followed a problem-based approach exemplifying those aspects of interest that could be verified on a transformation and discussing how they can be verified. The literature review can be found in Appendix A.

At this point it is clear that there exist several alternatives, not only for the specification of a transformation but also for its formal verification, which depend on those properties that must be addressed in each specific case. This problem increases when considering bidirectional, higher-order, and multi-model transformations. In this sense, there is some parallelism between MDE-built systems and traditional software systems: heterogeneous multi-logic specifications are needed, since any system has different aspects that are best specified in different semantic domains. An example of this was introduced in the last section, where a small-size example introduces different problems, each of them best addressed by several strategies.

To cope with this situation it is usually proposed a separation of duties between software developers. On the one side there are those experts in the MDE domain, and on the other, those in formal verification. This gives rise to different technological spaces

[KBA02], i.e. working contexts with a set of associated concepts, body of knowledge, tools, required skills, and possibilities. In general terms, MDE experts define models and transformations, as well as formal verification experts conduct the verification process, often aided by some (semi)automatic generation process which translates the MDE elements to their formal representation in the semantic domain used by the experts for verification purposes.

Related to this, the most common transformation approaches referred in the literature are the relational and graph-based approaches. The point here is that those approaches closely related with traditional programming languages (direct manipulation, operational, etc.) introduce verification problems that are carried out by traditional code verification approaches. Moreover, both relational and graph-based approaches define elements that are most easily translated into formal domains preserving its semantics.

The landscape before us leads us to think about trying a heterogeneous strategy to verification, closely related to the ideas in [CKTW08,Mos05]. In these works the authors define an environment where specification languages are described in their "natural" semantics, and the relations between languages are expressed by appropriate translations. Moreover, different semantic domains have associated different tools for formal reasoning, and since there are formal translations between languages, it is possible to "share" tools. These ideas are based in the Theory of Institutions [GB83].

Working on this idea, we could express metamodels and transformations as so called *institutions* in some consistent and interdependent way, as well as transformation properties in different logics (also institutions), and their translations (formally, institution (co)morphisms) into several logics with the purpose of proving transformation properties. Models could be both represented as institutions or as sentences within the institution of metamodels. For example, consider the Statecharts to Petri Nets example in the last section. We can represent models in XMI, metamodels in MOF, and transformations in QVT. These languages could be defined as institutions and there could be translations from them to different logics (also specified as institutions), e.g. FOL, Higher-Order Logic (HOL), rewriting logic, modal logic, etc. If there is a conformance need specified in OCL, it could be possible to translate the different elements to rewriting logic and perform the verification as defined in [**?**].

To put these ideas into practice we can use the Heterogeneous Tool Set (Hets) [Mos05,MML07] which is meant to support heterogeneous multi-logic specifications. Hets is a parsing, static analysis and proof management tool combining various such tools for individual specification languages. Nowadays the tool supports many logics (e.g. FOL, HOL, rewriting logic and modal logic) and tools (e.g. Isabelle/HOL and Maude). Moreover, it provides proof management capabilities for monitoring the overall correctness of a heterogeneous specification whereas different parts of it are verified using, possibly different, proof systems.

Anyway, it is worth pointing out that the instantiation of this framework is not as direct as it seems. We need to formally specify firstly every MDE building block within the Theory of Institutions and secondly any possible and useful translation to those logics we need. These representations may neither be direct nor even possible within this theory.

Besides, any new logic in Hets may support formal verification in its own semantic domain. However, this requires the definition of a sound proof system. In some cases where there is no such proof system, its definition can be very expensive. For example, in [LMAL10] the authors introduce a language for assertions and a semi-automated reasoning system which is not formally specified as a sound proof system.

A Annotated Bibliography

We include here a comprehensive annotated bibliography on verification of model transformations. The analysis is focused on summarizing the contributions of each paper and on identifying the transformation approach involved, the properties verified and the verification approach followed.

[ABGR10,ABK07] The authors introduce a method for representing MOF metamodels enriched with well-formedness rules expressed in OCL and declarative model transformations in a formalism called Alloy, based on first-order logic. They also show how first-order formulas are transformed into boolean expressions and then analyzed by the Alloy Analyzer which embeds different SAT solvers. Two kinds of properties are verified. First the analyzer tries to produce an instance of the source metamodel, and if the analyzer cannot produce it, there is an inconsistency in the definition of the transformation rules (Preservation of Execution Semantics) Second, the analyzer can check well-formedness of models w.r.t. metamodels (Conformance and Model Typing).

[AK02,AKP03] The authors propose an approach for expressing structural relations (using OCL) between two metamodels enriched with OCL invariants. The specification is then transformed into Java code and use OCL4Java, a Java library which allows dynamic evaluation of OCL constraints. The tool can check well-formedness of models w.r.t. metamodels (Conformance and Model Typing) and also allows checking the satisfiability of structural relations (Model Syntax Relations)

[ALL09,ALL10,AMV$^+$10,LMAL10] The authors develop a semi-automated reasoning system that can prove some properties of QVT relational and graph-based transformations in an offline way. They use the idea of an assertion (a.k.a. transformation contract or verification specification): a formal expression in first-order logic) that describes some characteristics of a model (Conformance and Model Typing) under transformation or a model transformation (Model Syntax Relations) They devise a (limited) language for assertions. A transformation description with basic assertions describing the rewriting rules can be generated automatically (using the Visual Modeling and Transformation System (VMTS) tool) from the definition of any transformation. For the reasoning system, they use SWI-Prolog. The engine defines how an assertion evolves when applying a transformation rule. When every rule has been applied the engine constructs a final assertion. If the assertions to be verified can be derived from the final assertion, they infer that they hold in the target model.

[BBG$^+$06] The authors present a verification technique for arbitrarily large multi-agent systems featuring complex coordination and structural adaptation. They verify that none of the structural adaptations within the system leads to an unsafe system state (Model Semantics Relations) The system state is modeled as a graph, system transitions are modeled as rule applications in a graph transformation system, and safety properties of the system are encoded as inductive invariants. They use symbolic encodings of graph patterns into first-order predicate logic by using the relational programming language Relational Meta-Language (RML). Finally they use the CrocoPat interpreter for that language for performing the verification.

[BEH06] The authors define a relation of semantic compatibility between processes by requiring weakly bisimilarity (Model Semantics Relations) They use a graph transformation system specifying the operational semantics of the transformation between processes. The verification is performed by critical pair analysis supported by the AGG tool.

[BHM09] The authors present an approach for formalizing and verifying QVT-like transformations that reuses the main concepts of graph transformation systems. Specifically, they formalize model transformations as theories in rewriting logic, so that Maude's reachability analysis and model checking features can be used for verifying temporal properties (Model Semantics Relations)

[BKMW08] The authors work with functional model transformations (not in any particular language) where a transformation can be specified as a set of recursive model equations, which are automatically translated into ordinary Membership Equational Logic (MEL) equations in the MOdel manageMENT (MOMENT) framework (based on Maude). The correctness of a transformation is stated in terms of the existence of an intermediate institution that relates the source and target institutions through a span of an institution morphism and an institution comorphism. They do not verify particular properties but somehow with the construction of the intermediate institution they are proving that the models are in some sense semantically equivalent (Model Semantics Relations).

[BLA$^+$10] The authors present a Turing incomplete language and a tool for model transformations (DSLTrans). The language, based on typed graphs, guarantees termination (Termination) and confluence (Determinism) by construction.

[Bru08] The author describes a method for proving the termination (Termination) of graph transformation systems. He develops an algorithm (at a theoretical level) which can prove termination but with some problems that are left to future study.

[CBBD09] The authors propose a method for verifying that the result of a transformation is correct with respect to the transformation specification. This is achieved by verifying transformation contracts written in OCL, partially or fully generated through a dedicated tool. A transformation contract is composed by invariants on the source/target model (Conformance and Model Typing) and constraints on element evolution (Model Syntax Relations) They use an OCL-checker to verify the properties. They do not use any particular transformation approach. In fact, they assume that the transformation is performed by someone else and they just focus on proving whether the contract is satisfied or not. Finally, they develop an example where they verify a refinement property (Model Semantics Relations)

[CCGdL09,BCG11] The authors propose a method for constructing a transformation model, i.e. a model composed by the source and target metamodel, the transformation specification and the well-formedness rules. Transformations are specified in a declarative way using either QVT or Triple Graph Grammars (TGG). They define a number of invariant-based verification properties which can be derived from the transformation model. The properties are expressed at two levels: considering the role of individual rules within a transformation, or considering the transformation model as a whole. For the first level they define the following properties: applicable (Functional Behavior), (weak/strong)executable (Functional Behavior), total (Syntactic Completeness), deterministic (Determinism + Functional Behavior), functional (Determinism + Functional Behavior), exhaustive (surjective) (Functional Behavior), injective (Functional Behavior), bijective (Functional Behavior), redundant (Preservation of Execution Semantics), enabledness subsumption (Preservation of Execution Semantics). For the second, they define the following ones: executable (Functional Behavior), total (Syntactic Completeness), deterministic (Determinism + Functional Behavior), functional (Determinism + Functional Behavior), exhaustive (Functional Behavior), injective (Functional Behavior), bijective (Functional Behavior). All these properties can be encoded as UML/OCL consistency problems on the transformation model. They use the tool UMLtoCSP which translates the transformation model into a constraint satisfaction problem and then processed with constraint solvers (with the problem of the search space dimension). The second paper can be seen as an extension of this approach to ATL.

[CCGT09] The authors formally prove the weak simulation (Model Semantics Relations) of xSPEM models transformed (using ATL) into Petri Nets. This relation ensures that the conclusions obtained on the Petri Nets also hold on the xSPEM model.

[Cha06] The authors describe an approach for producing formal proofs, based on the proof-as-programs methodology (see [Poe08]), for a particular QoS-oriented transformational system. However, they do not give enough details about neither what properties they prove nor what tools they use.

[CHM$^+$02] The authors present the VIATRA framework, a transformation-based verification and validation environment. Syntactic correctness (Model Syntax Relations) and completeness (Syntactic Completeness) can be verified by planner algorithms. On the other hand, semantic correctness (Model Semantics Relations) of transformations is verified by projecting transformation rules into the Symbolic Analysis Laboratory (SAL) intermediate language, which provides access to an automated combination of symbolic analysis tools (like model checkers and theorem provers). They claim to have verified safety properties but they give no details.

[CLST10] The authors present a framework based on the Calculus of Inductive Constructions (CIC) and its associated tool the Coq proof assistant to allow certification of relational model transformations. The approach is based on a semi-automatic translation process from metamodels, models and transformations into types, propositions and functions in CIC. Then, the verification is interactively performed in Coq by proving the postconditions assuming that both the pre-conditions and the transformation rules hold. The approach allows verification of any kind of property, both on metamodels (Conformance and Model Typing) and transformations (Model Syntax Relations and Model Semantic Relations)

[CNS12] The authors state several properties that merge operators (N-Ary Transformations Prop.) should possess: completeness means no information is lost during the merge; non-redundancy ensures that the merge does not duplicate redundant information spread over source models; minimality ensures that the merge produces models with information solely originated from sources; totality ensures that the merge can actually be computed on any pair of models; idempotency ensures that a model merged with itself produces an identical model. Beyond merging, they can potentially characterize other transformations, not necessarily involving several source models. They implement merge operators in the TReMer+ tool which provides support for checking the consistency of the relationships and the merged models by translating the information to first order logic and using the relational checker (CrocoPat) for its verification.

[EKHL03] The authors present in an abstract way how verification and validation (of general properties) can be achieved for UML models. Within the approach, graph transformation techniques are applied for automated translation of UML models into a language understood by a verification tool or directly into an implementation. In particular they exemplify the approach by proving the property of deadlock freedom (Model Semantics Relations) on a UML system model represented in UML-RT. Such a system model has been partially translated into Communicating Sequential Processes (CSP) and a condition for deadlock freedom has been stated. Finally the property is verified using a model checker.

[EEdL$^+$05] The authors present an approach for proving the termination (Termination) of a graph transformation by performing critical pair analysis using the AGG tool.

[GGL$^+$06] The authors verify that a given graph-based transformation (specified using TGG) ensures semantic equivalence (Model Semantics Relations) between a source model in a given model specification language and the resulting programming language code. The correctness of this specification is shown using the theorem prover Isabelle/HOL.

[GdLW$^+$12] The authors propose a declarative language for the specification of visual contracts, enabling the verification of relational transformations defined with any transformation language. The verification is performed by compiling the contracts into QVT to detect disconformities of transformation results (model level) with respect to the contracts (Model Syntax Relations) The language allows for reasoning on: metamodel coverage (Syntactic Completeness), redundancies, contradictions and pattern satisfaction on contracts (Preservation of Execution Semantics)

[GM07] The authors propose the use of +CAL, a specification language designed to replace pseudo-code for writing high-level descriptions of algorithms, for representing metamodels and invariants, as well as mapping transformation specifications (with no specific approach) into the +CAL model-checking language. With this approach they can verify properties on source and target models (Conformance and Model Typing) as well as check the satisfaction of contracts on the transformation (Model Syntax Relations).

[HEOG10] The authors analyze local confluence (Determinism) in the view of functional behavior for graph-based model transformations specified with TGG and executed in the AGG tool. In this context, functional behavior means that for each graph in the source language the transformation yields a unique graph (up to isomorphism) in the target language. Local confluence is proved by computing critical pairs using the AGG tool.

[HHK10] The authors state different aspects to be verified within a model transformation and focus on verifying graph-based model transformations (specified using TGG) with respect to behavioral equivalence (Model Semantics Relations) They do not implement the approach.

[HKT02] The authors formally prove the (local) confluence (Determinism) for typed attributed graph transformations by computing critical pairs using the AGG tool.

[HT05,MGB05,RLK$^+$08] These three works present how behavioral preservation (Model Semantics Relations) can be verified for refactoring transformations specified using a graph-based approach. The first paper discusses semantic conditions for the refinement of dynamic models expressed as instances of graph transformation systems. The second proposes semantics-preserving transformations (for no specific transformation approach) for UML class diagrams annotated with OCL that are proven sound according to a translational semantics in Alloy. Finally, the third article uses a bisimulation checking algorithm for borrowed contexts providing the means for automatically checking models for behavior preservation.

[Kat06] The author study temporal logics to characterize properties of weaved aspects (N-Ary Transformations Properties) Static analysis techniques enriched with dataflow information or richer type systems are generally used to detect these properties.

[KMS^{+}09] The authors studied the available alternatives for explicit modeling of transformations for the detection of syntactic errors (Typing) in a model transformation (in any language): either using a dedicated metamodel as a basis for deriving a specialized transformation language, or directly using the original metamodel and then modulating the conformance checking accordingly, for deriving such a language.

[Küs04,Küs06] In the first article the author introduce different properties that must be verified in a graph-based model transformation and then focuses on validation of syntactic correctness (Conformance and Model Typing) for rule-based model transformations. In the second article he concentrates on proving termination criteria (Termination) and (local) confluence (Determinism) properties to be checked at design time by static analysis of the transformation rules and the underlying metamodels.

[LBA10] The authors formally define a property language to express structural relations between two language's metamodels (Model Syntax Relations) and propose a symbolic technique (based on model-checking) to verify whether those relations hold, given input and output metamodels, and a graph-based transformation. The symbolic technique is implemented in SWI-Prolog.

[Lan06,LR10] The authors formalize metamodels, models and transformations in UML Reactive System Development Support (UML-RSDS) (a subset of UML with a precise axiomatic semantics). The language allows the translation of this information into the B language and thus B provers are used to analyze and prove the correctness of the following properties: determinacy/uniqueness (Determinism); confluence (Confluence); definedness, i.e. whether the transformation is applicable to every model of the source language (Syntactic Completeness); syntactic correctness, i.e. whether a transformation maps correct models of the source language into correct models of the target language (Conformance and Model Typing); rule completeness, i.e. whether the intended effect of a transformation rule can be unambiguously deduced from its explicit specification (Model Syntax Relations); and language-level semantic correctness, i.e. whether there is an interpretation from the source language to the target language such that, for any models semantically conforming to the source language, the interpretation semantically conforms to the target language (Model Semantics Relations).

[LD10] The authors formalize metamodels, models and transformations (in any specific model transformation language) in B. Then the B provers are used to analyze and prove the correctness of transformation rules. This schema is used to prove correctness of transformation rules w.r.t. metamodels (Conformance and Model Typing) and transformation invariants (Model Syntax Relations), as well as for proving the consistency of transformation rules w.r.t. each other (Preservation of Execution Semantics).

[LEO06,Bie11] The first work proposes an efficient method to compute the set of all critical pairs of a given graph transformation system intended to prove that a graph transformation system is locally confluent (Determinism). For the critical pair analysis, they use the AGG tool. The second one extends the results of the former for the EMF. They also study consistency properties between EMF transformation rules (Preservation of Execution Semantics)

[LLM09] The authors propose an approach based on domain-specific design patterns together with a relaxed conformance relation to allow the use of model fragments to construct any model instance from a given metamodel (Typing).

[LR11] The authors define different correctness criteria for model transformations (not in some specific language): representation (Conformance and Model Typing), syntactic correctness (Conformance and Model Typing), and language-level and model-level semantic correctness (Model Semantics Relations) However, since the paper is focused on presenting a systematic model-driven development process for model transformations, they do not present how the verification can be done.

[MSJ$^+$10] The authors present a language for Multi-Dimensional Separation of Concerns implemented in the AGG tool. This framework allows detecting conflicts between aspects (N-Ary Transformations Prop.) by model-checking, typically when multiple advices must be executed on the same join points.

[NK08b,NK08a,NK08c] The first paper presents a method for the verification of semantic equivalence (Model Semantics Relations) for graph-based transformations. In particular they prove that a transformation outputs a statechart bisimilar to an input statechart. The second and third paper also present a language for defining structural correspondence between metamodels (Model Syntax Relations). They also present an extension of the transformation so that these rules can be model-checked on the instance models. The verification is performed at the instance level using the GReAT tool.

[OW09] The authors present a method to specify the properties that characterize the correctness of a given pattern-based model-to-model transformation (closely related to relational model transformations), and they develop the basis for a method for checking that a transformation is correct. Metamodels are represented as algebras, transformations as triple patterns and verification properties as finite sets of triple algebraic patterns. Then a verification specification (a.k.a. transformation contract) can represent constraints on element evolution (Model Syntax Relations) After the transformation is executed both transformation rules and verification specification must hold. Finally they devise two options for the implementation of this approach: either using a tool like Maude that would allow them to directly work on algebras, or specializing the approach to the case of graphs and using some graph transformation tool.

[PCG11] The authors propose a method that performs a static analysis of the relational ATL rules with respect to two correctness properties: weak executability (Preservation of Execution Semantics), which determines if there is some scenario in which an ATL rule can be safely applied without breaking the target metamodel integrity constraints; and coverage (Syntactic Completeness), which ensures a set of ATL rules allow addressing all elements of the source and target metamodels.

[PG08] The authors describe a lightweight validation approach for checking refinement conditions (Model Semantics Relations) for UML class diagrams. Refinement is not explicitly defined as a model transformation. Refinement properties are specified in OCL and it verification is performed by using a hybrid strategy that combines model checking, testing and theorem proving.

[PGE97] The authors introduce a morphism that allows preserving safety properties (Model Semantics Relations) of an algebraic Petri Net when the net is structurally modified. Although this work was not explicitly created for the purpose of model transformation verification, it could be used to generate a set of model transformations that would preserve invariants in algebraic petri nets by construction.

[Poe08,PT10] In the first paper the author outline how Constructive Type Theory can be used to provide a uniform formal foundation for representing models, metamodels and model transformations (relational, but using any specific language). Then properties of any kind on metamodels (Conformance and Model Typing) and on transformations (Model Syntax Relations and Model Semantic Relations) could be verified using a interactive theorem prover. Moreover, given a proof in this kind, the proof-as-programs approach can be applied to extract a correct model transformation. The second paper presents how this approach could be implemented by using the Coq proof assistant.

[Sch10] The authors present how models, metamodels and declarative transformation rules, as well as associated soundness conditions, can be directly translated into Isabelle/HOL. Formal verification of general properties on metamodels (Conformance and Model Typing) and on transformations (Model Syntax Relations and Model Semantic Relations) can be performed using the theorem prover. In particular, the approach is demonstrated for a refactoring transformation and a related soundness condition.

[SJ07] The authors propose a simple extension to object-oriented typing to better cater for a model-oriented context, including a simple strategy for typing models as a collection of interconnected objects. They suggest extensions to existing type system formalisms to support these concepts and their manipulation.

[SMR11] The authors present a formal calculus for operational QVT. The calculus is implemented in the interactive theorem prover KIV (an interactive theorem prover based on algebraic specifications with several logical extensions, e.g. Dynamic Logic for imperative programs and Java and temporal logic for parallel programs and state charts) and allows to prove properties of QVT transformations for arbitrary meta models. In particular, it is possible to formally prove that a transformation always generates a Java model that is type correct (Conformance and Model Typing) and has certain semantic properties (Model Semantics Relations).

[VP03] The authors present an automated technique to formally verify by model checking that a graph-based model transformation preserves dynamic consistency properties (Model Semantics Relations) First a behaviorally equivalent transition system is generated automatically for both the source and the target model. Then they define one or more semantic properties in the source language (potentially using temporal logic operators) which are transformed into semantically equivalent properties in the target language (manually, or using the same transformation program). Finally both transition systems are model-checked automatically (e.g. using the SAL tool) to prove the properties. If the verification succeeds, then it is possible to conclude that the model transformation is correct with respect to the pair of properties for the specific pairs of source and target models.

[VR11] The authors propose a static analyzer for inspecting the source code of ATL transformations (declarative and imperative). It provides an API comprising methods to extract and handle diverse elements from ATL transformations. Although the API could be used to the verification of a transformation, they do not introduce any particular problem nor any formalism to it verification.

[VVGE+06] The authors propose a Petri Net based analysis method that provides a sufficient criterion for the termination (Termination) problem of model transformations captured by graph transformation systems.

[WKK+09] The authors present a framework (TRansformations On Petri Nets In Color (TROPIC)) where a relational transformation is expressed as a CPN which allows for the use of existing CPN execution engines to simulate transformations and the formal exploration of CPN properties. They also define a taxonomy of transformation errors and CPN properties related to these errors, mostly focused on proving language-related properties (Conformance and Model Typing, Preservation of Execution Semantics, Termination, Determinism) but for a given model transformation.

[BDTM+06,BFS+06,BGF+10,BK11,BKE11,DPV08,FBMLT09,FMOP10]
[FSB04,GV11,KAER06,LZG05,MBT06b,MBT06a,SBM09,WKC06] Although we are interested in formal verification of model transformations, we reviewed some works on testing of model transformations with the purpose of indentifying potential verification properties addressed by them. For a comprehensive review on this topic the reader may refer to [LPCL11]. The works mentioned above, do not address specific verification problems, except for some of them which talk about

metamodel coverage, e.g.[KAER06,WKC06]. In general they address the problem of generating appropriate test cases and how different traditional testing techniques could be adapted to verify model transformations. Although testing cannot prove the correctness of a model transformation, any technique could be used to disprove it correctness and to increase the confidence level on any kind of expected property.

References about Verification of Model Transformations

ABGR10. Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. On challenges of model transformation from UML to Alloy. *Software and Systems Modeling*, 9:69–86, 2010.

ABK07. Kyriakos Anastasakis, Behzad Bordbar, and Jochen M. Küster. Analysis of model transformations via Alloy. In Benoit Baudry, Alain Faivre, Sudipto Ghosh, and Alexander Pretschner, editors, *Proceedings of the 4th MoDeVVa workshop Model-Driven Engineering, Verification and Validation*, pages 47–56, 2007.

AK02. David H. Akehurst and Stuart Kent. A relational approach to defining transformations in a metamodel. In Jean-Marc Jézéquel, Heinrich Hußmann, and Stephen Cook, editors, *UML*, volume 2460 of *Lecture Notes in Computer Science*, pages 243–258. Springer, 2002.

AKP03. David Akehurst, Stuart Kent, and Octavian Patrascoiu. A relational approach to defining and implementing transformations between metamodels. *Software and System Modeling*, 2(4):215–239, 2003.

ALL09. Márk Asztalos, László Lengyel, and Tihamér Levendovszky. A formalism for describing modeling transformations for verification. In *Proceedings of the Model-Driven Engineering, Verification, And Validation (MoDEVa) workshop*, 2009.

ALL10. Márk Asztalos, László Lengyel, and Tihamer Levendovszky. Towards automated, formal verification of model transformations. In *ICST*, pages 15–24. IEEE Computer Society, 2010.

ALS$^+$12. Moussa Amrani, Levi Lucio, Gehan Selim, Benoit Combemale, Jurgen Dingel, Hans Vangheluwe, Yves Le traon, and James Cordy. A tridimensional approach for studying the formal verification of model transformations. In *Verification and validation Of model Transformations (VOLT)*. IEEE, 2012.

AMV$^+$10. Mark Asztalos, Istvan Madari, Tamas Vajk, Laszlo Lengyel, and Tihamer Levendovszky. Formal verification of model transformations: An automated framework. In *Computational Cybernetics and Technical Informatics (ICCC-CONTI), 2010 International Joint Conference on*, pages 493 –498, may 2010.

BBG$^+$06. Basil Becker, Dirk Beyer, Holger Giese, Florian Klein, and Daniela Schilling. Symbolic invariant verification for systems with dynamic structural adaptation. In Leon J. Osterweil, H. Dieter Rombach, and Mary Lou Soffa, editors, *ICSE*, pages 72–81. ACM, 2006.

BCG11. Fabian Büttner, Jordi Cabot, and Martin Gogolla. On validation of ATL transformation rules by transformation models. In *Proceedings of the 8th International Workshop on Model-Driven Engineering, Verification and Validation (MoDeVVa)*, pages 9:1–9:8. ACM, 2011.

BDTM$^+$06. Benoit Baudry, Trung Dinh-Trong, Jean-Marie Mottu, Devon Simmonds, Robert France, Sudipto Ghosh, Franck Fleurey, and Yves Le Traon. Model transformation testing challenges. In *Proceedings of the IMDT workshop in conjunction with ECMDA 06*, 2006.

BEH06. Luciano Baresi, Karsten Ehrig, and Reiko Heckel. Verification of model transformations: A case study with BPEL. In Ugo Montanari, Donald Sannella, and Roberto Bruni, editors, *TGC*, volume 4661 of *Lecture Notes in Computer Science*, pages 183–199. Springer, 2006.

BFS$^+$06. Erwan Brottier, Franck Fleurey, Jim Steel, Benoit Baudry, and Yves Le Traon. Metamodel-based test generation for model transformations: an algorithm and a tool. In *ISSRE*, pages 85–94. IEEE Computer Society, 2006.

BGF$^+$10. Benoit Baudry, Sudipto Ghosh, Franck Fleurey, Robert France, Yves Le Traon, and Jean-Marie Mottu. Barriers to systematic model transformation testing. *Commun. ACM*, 53(6):139–143, June 2010.

BHM09. Artur Boronat, Reiko Heckel, and José Meseguer. Rewriting logic semantics and verification of model transformations. In *Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering (FASE'09)*, pages 18–33. Springer, 2009.

Bie11. Enrico Biermann. Local confluence analysis of consistent EMF transformations. *ECEASST*, 38, 2011.

BK11. Eduard Bauer and Jochen Malte Küster. Combining specification-based and code-based coverage for model transformation chains. In Cabot and Visser [CV11], pages 78–92.

BKE11. Eduard Bauer, Jochen Malte Küster, and Gregor Engels. Test suite quality for model transformation chains. In Judith Bishop and Antonio Vallecillo, editors, *TOOLS (49)*, volume 6705 of *Lecture Notes in Computer Science*, pages 3–19. Springer, 2011.

BKMW08. Artur Boronat, Alexander Knapp, José Meseguer, and Martin Wirsing. What is a multi-modeling language? In Andrea Corradini and Ugo Montanari, editors, *WADT*, volume 5486 of *Lecture Notes in Computer Science*, pages 71–87. Springer, 2008.

BLA$^+$10. Bruno Barroca, Levi Lucio, Vasco Amaral, Roberto Félix, and Vasco Sousa. DSLTrans: A turing incomplete transformation language. In Brian A. Malloy, Steffen Staab, and Mark van den Brand, editors, *SLE*, volume 6563 of *Lecture Notes in Computer Science*, pages 296–305. Springer, 2010.

Bru08. H.J. Sander Bruggink. Towards a systematic method for proving termination of graph transformation systems. *Electron. Notes Theor. Comput. Sci.*, 213(1):23–38, May 2008.

CBBD09. Eric Cariou, Nicolas Belloir, Franck Barbier, and Nidal Djemam. OCL contracts for the verification of model transformations. *OCL workshop of MoDELS 2009*, 2009.

CCGdL09. Jordi Cabot, Robert Clarisó, Esther Guerra, and Juan de Lara. Verification and validation of declarative model-to-model transformations through invariants. *Journal of Systems and Software*, August 2009.

CCGT09. Benoît Combemale, Xavier Crégut, Pierre-Loïc Garoche, and Xavier Thirioux. Essay on semantics definition in MDE - an instrumented approach for model verification. *JSW*, 4(9):943–958, 2009.

Cha06. Kenneth Chan. Formal proofs for QoS-oriented transformations. In *EDOC Workshops*, page 41. IEEE Computer Society, 2006.

CHM$^+$02. György Csertán, Gábor Huszerl, István Majzik, Zsigmond Pap, András Pataricza, and Dániel Varró. VIATRA - visual automated transformations for formal verification and validation of uml models. In *ASE*, pages 267–270. IEEE Computer Society, 2002.

CLST10. Daniel Calegari, Carlos Luna, Nora Szasz, and Alvaro Tasistro. A type-theoretic framework for certified model transformations. In Jim Davies, Leila Silva, and Adenilso da Silva Simão, editors, *SBMF*, volume 6527 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2010.

CNS12. Marsha Chechik, Shiva Nejati, and Mehrdad Sabetzadeh. A relationship-based approach to model integration. *Innov. Syst. Softw. Eng.*, 8(1):3–18, March 2012.

CV11. Jordi Cabot and Eelco Visser, editors. *Theory and Practice of Model Transformations - 4th International Conference (ICMT 2011). Proceedings*, volume 6707 of *Lecture Notes in Computer Science*. Springer, 2011.

DPV08. Andrea Darabos, András Pataricza, and Dániel Varró. Towards testing the implementation of graph transformations. *Electr. Notes Theor. Comput. Sci.*, 211:75–85, 2008.

EEdL$^+$05. Hartmut Ehrig, Karsten Ehrig, Juan de Lara, Gabriele Taentzer, Dániel Varró, and Szilvia Varró-Gyapay. Termination criteria for model transformation. In Maura Cerioli, editor, *FASE*, volume 3442 of *Lecture Notes in Computer Science*, pages 49–63. Springer, 2005.

EHRT08. Hartmut Ehrig, Reiko Heckel, Grzegorz Rozenberg, and Gabriele Taentzer, editors. *Graph Transformations, 4th International Conference (ICGT 2008), UK. Proceedings*, volume 5214 of *Lecture Notes in Computer Science*. Springer, 2008.

EKHL03. Gregor Engels, Jochen Küster, Reiko Heckel, and Marc Lohmann. Model-based verification and validation of properties. *Electr. Notes Theor. Comput. Sci.*, 82(7), 2003.

FBMLT09. Franck Fleurey, Benoit Baudry, Pierre-Alain Muller, and Yves Le Traon. Towards dependable model transformations: Qualifying input test data. *Journal of Software and Systems Modeling (SoSyM)*, 8(2):185–203, 2009.

FMOP10. Camillo Fiorentini, Alberto Momigliano, Mario Ornaghi, and Iman Poernomo. A constructive approach to testing model transformations. In Laurence Tratt and Martin Gogolla, editors, *Theory and Practice of Model Transformations*, volume 6142 of *Lecture Notes in Computer Science*, pages 77–92. Springer, 2010.

FSB04. F. Fleurey, J. Steel, and B. Baudry. Validation in model-driven engineering: Testing model transformations. In *Proceedings of the 1st International Workshop on Model, Design and Validation*, pages 29–40, 2004.

GdLW$^+$12. Esther Guerra, Juan de Lara, Manuel Wimmer, Gerti Kappel, Angelika Kusel, Werner Retschitzegger, Johannes Schonbock, and Wieland Schwinger. Automated verification of model transformations based on visual contracts. *Automated Software Engineering*, pages 1–42, 2012.

GGL$^+$06. Holger Giese, Sabine Glesner, Johannes Leitner, Wilhelm Schäfer, and Robert Wagner. Towards verified model transformations. In David Hearnden, Jörn Guy Süß, Benoit Baudry, and Nicolas Rapin, editors, *Proceedings of the 3rd International Workshop on Model Development, Validation and Verification, Italy*, pages 78–93. Le Commissariat a l'Energie Atomique - CEA, October 2006.

GM07. Miguel Garcia and Ralf Möller. Certification of transformations algorithms in model-driven software development. In Wolf-Gideon Bleek, Jorg Räsch, and Heinz Züllighoven, editors, *Software Engineering 2007*, volume 105 of *GI-Edition Lecture Notes in Informatics*, pages 107–118, 2007.

GV11. Martin Gogolla and Antonio Vallecillo. *Tract*able model transformation testing. In Robert B. France, Jochen Malte Küster, Behzad Bordbar, and Richard F. Paige, editors, *ECMFA*, volume 6698 of *Lecture Notes in Computer Science*, pages 221–235. Springer, 2011.

HEOG10. Frank Hermann, Hartmut Ehrig, Fernando Orejas, and Ulrike Golas. Formal analysis of functional behaviour for model transformations based on triple graph grammars. In Hartmut Ehrig, Arend Rensink, Grzegorz Rozenberg, and Andy Schürr, editors, *ICGT*, volume 6372 of *Lecture Notes in Computer Science*, pages 155–170. Springer, 2010.

HHK10. Frank Hermann, Mathias Hülsbusch, and Barbara König. Specification and verification of model transformations. *ECEASST*, 30, 2010.

HKT02. Reiko Heckel, Jochen Malte Küster, and Gabriele Taentzer. Confluence of typed attributed graph transformation systems. In Andrea Corradini, Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *ICGT*, volume 2505 of *Lecture Notes in Computer Science*, pages 161–176. Springer, 2002.

HT05. Reiko Heckel and Sebastian Thöne. Behavioral refinement of graph transformation-based models. *Electr. Notes Theor. Comput. Sci.*, 127(3):101–111, 2005.

KAER06. Jochen Malte Küster and Mohamed Abd-El-Razik. Validation of model transformations - first experiences using a white box approach. In Thomas Kühne, editor, *MoDELS Workshops*, volume 4364 of *Lecture Notes in Computer Science*, pages 193–204. Springer, 2006.

Kat06. Shmuel Katz. Aspect categories and classes of temporal properties. In Awais Rashid and Mehmet Aksit, editors, *T. Aspect-Oriented Software Development I*, volume 3880 of *Lecture Notes in Computer Science*, pages 106–134. Springer, 2006.

KMS$^+$09. Thomas Kühne, Gergely Mezei, Eugene Syriani, Hans Vangheluwe, and Manuel Wimmer. Systematic transformation development. *ECEASST*, 21, 2009.

Küs04. J. M. Küster. Systematic validation of model transformations. In *Proceedings of WiSME'04 (associated to UML'04)*, 2004.

Küs06. Jochen Küster. Definition and validation of model transformations. *Software and System Modeling*, 5(3):233–259, 2006.

Lan06. Kevin Lano. Using B to verify UML transformations. *MODEVA workshop at MoDELS 2006*, 2006.

LBA10. Levi Lucio, Bruno Barroca, and Vasco Amaral. A technique for automatic validation of model transformations. In Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen, editors, *MoDELS (1)*, volume 6394 of *Lecture Notes in Computer Science*, pages 136–150. Springer, 2010.

LD10. Hung Ledang and Hubert Dubois. Proving model transformations. In Jing Liu, Doron Peled, Bow-Yaw Wang, and Farn Wang, editors, *TASE*, pages 35–44. IEEE Computer Society, 2010.

LEO06. Leen Lambers, Hartmut Ehrig, and Fernando Orejas. Efficient detection of conflicts in graph-based model transformation. *Electron. Notes Theor. Comput. Sci.*, 152:97–109, March 2006.

LLM09. Tihamer Levendovszky, László Lengyel, and Tamás Mészáros. Supporting domain-specific model patterns with metamodeling. *Software and System Modeling*, 8(4):501–520, 2009.

LMAL10. Laszlo Lengyel, Istvan Madari, Mark Asztalos, and Tihamer Levendovszky. Validating Query/View/Transformation relations. *Model-Driven Engineering, Verification, and Validation, Workshop on*, 0:7–12, 2010.

LPCL11. Leonardo Lopez, Leonardo Pintos, Daniel Calegari, and Carlos Luna. Estado del arte de testing de transformaciones de modelos. Technical Report 11-01, InCo-PEDECIBA, 2011.

LR10. Kevin Lano and Shekoufeh Kolahdouz Rahimi. Specification and verification of model transformations using UML-RSDS. In Dominique Méry and Stephan Merz, editors, *IFM*, volume 6396 of *Lecture Notes in Computer Science*, pages 199–214. Springer, 2010.

LR11. Kevin Lano and Shekoufeh Kolahdouz Rahimi. Model-driven development of model transformations. In Cabot and Visser [CV11], pages 47–61.

LVV$^+$10. Horacio Lopez, Fernando Varesi, Marcelo Viniolo, Daniel Calegari, and Carlos Luna. Estado del arte de verificacion de transformaciones de modelos. Technical Report 10-07, InCo-PEDECIBA, 2010.

LZG05. Yuehua Lin, Jing Zhang, and Jeff Gray. A testing framework for model transformations. *Model-Driven Software Development - Research and Practice in Software Engineering. 2005*, pages 219–236, 2005.

MBT06a. Jean-Marie Mottu, Benoit Baudry, and Yves Le Traon. Mutation analysis testing for model transformations. In Arend Rensink and Jos Warmer, editors, *ECMDA-*

FA, volume 4066 of *Lecture Notes in Computer Science*, pages 376–390. Springer, 2006.

MBT06b. Jean-Marie Mottu, Benoit Baudry, and Yves Le Traon. Reusable MDA components: A testing-for-trust approach. In Oscar Nierstrasz, Jon Whittle, David Harel, and Gianna Reggio, editors, *MoDELS*, volume 4199 of *Lecture Notes in Computer Science*, pages 589–603. Springer, 2006.

MGB05. Tiago Massoni, Rohit Gheyi, and Paulo Borba. Formal refactoring for UML class diagrams. In *19th brazilian Symposium on Software Engineering (SBES)*, pages 152–167, 2005.

MSJ$^+$10. Tim Molderez, Hans Schippers, Dirk Janssens, Michael Haupt, and Robert Hirschfeld. A platform for experimenting with language constructs for modularizing crosscutting concerns. In *Proceedings of the 3rd International Workshop on Academic Software Development Tools and Techniques (WASDeTT)*, 2010.

NK08a. Anantha Narayanan and Gabor Karsai. Specifying the correctness properties of model transformations. In *Proceedings of the third international workshop on Graph and model transformations (GRaMoT'08)*, pages 45–52. ACM, 2008.

NK08b. Anantha Narayanan and Gabor Karsai. Towards verifying model transformations. *Electr. Notes Theor. Comput. Sci.*, 211:191–200, 2008.

NK08c. Anantha Narayanan and Gabor Karsai. Verifying model transformations by structural correspondence. *ECEASST*, 10, 2008.

OW09. Fernando Orejas and Martin Wirsing. On the specification and verification of model transformations. In Jens Palsberg, editor, *Semantics and Algebraic Specification*, volume 5700 of *Lecture Notes in Computer Science*, pages 140–161. Springer, 2009.

PCG11. Elena Planas, Jordi Cabot, and Cristina Gómez. Two basic correctness properties for ATL transformations: Executability and coverage. In *3rd International Workshop on Model Transformation with ATL*, Zurich, Switzerland, July 2011.

PG08. Claudia Pons and Diego Garcia. A lightweight approach for the semantic validation of model refinements. *Electronic Notes in Theoretical Computer Science*, 2008.

PGE97. J. Padberg, M. Gajewsky, and C. Ermel. Refinement versus verification: Compatibility of net-invariants and stepwise development of high-level petri nets. Technical Report 97-22, Technical University Berlin, 1997.

Poe08. Iman Poernomo. Proofs-as-model-transformations. In Antonio Vallecillo, Jeff Gray, and Alfonso Pierantonio, editors, *ICMT*, volume 5063 of *Lecture Notes in Computer Science*, pages 214–228. Springer, 2008.

PT10. Iman Poernomo and Jeffrey Terrell. Correct-by-construction model transformations from partially ordered specifications in Coq. In Jin Song Dong and Huibiao Zhu, editors, *ICFEM*, volume 6447 of *Lecture Notes in Computer Science*, pages 56–73. Springer, 2010.

RLK$^+$08. Guilherme Rangel, Leen Lambers, Barbara König, Hartmut Ehrig, and Paolo Baldan. Behavior preservation in model refactoring using dpo transformations with borrowed contexts. In Ehrig et al. [EHRT08], pages 242–256.

SBM09. Sagar Sen, Benoit Baudry, and Jean-Marie Mottu. Automatic model generation strategies for model transformation testing. In Richard F. Paige, editor, *ICMT*, volume 5563 of *Lecture Notes in Computer Science*, pages 148–164. Springer, 2009.

Sch10. Bernhard Schätz. Verification of model transformations. *ECEASST*, 29, 2010.

SJ07. Jim Steel and Jean M. Jézéquel. On model typing. *Software and Systems Modeling*, 6(4):401–413, December 2007.

SK08. Andy Schürr and Felix Klar. 15 years of triple graph grammars. In Ehrig et al. [EHRT08], pages 411–425.

SMR11. Kurt Stenzel, Nina Moebius, and Wolfgang Reif. Formal verification of QVT transformations for code generation. In Jon Whittle, Tony Clark, and Thomas Kühne, editors, *MoDELS*, volume 6981 of *Lecture Notes in Computer Science*, pages 533–547. Springer, 2011.

VP03. Dániel Varró and András Pataricza. Automated formal verification of model transformations. In Jan Jürjens, Bernhard Rumpe, Robert France, and Eduardo B. Fernandez, editors, *Critical Systems Development in UML (CSDUML 2003), Proceedings of the UML'03 Workshop*, number TUM-I0323 in Technical Report, pages 63–78. Technische Universität München, September 2003.

VR11. Andreza Vieira and Franklin Ramalho. A static analyzer for model transformations. In *3rd International Workshop on Model Transformation with ATL*, Zurich, Switzerland, July 2011.

VVGE$^+$06. Dániel Varró, Szilvia Varró-Gyapay, Hartmut Ehrig, Ulrike Prange, and Gabriele Taentzer. Termination analysis of model transformations by petri nets. In Andrea Corradini, Hartmut Ehrig, Ugo Montanari, Leila Ribeiro, and Grzegorz Rozenberg, editors, *ICGT*, volume 4178 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2006.

WKC06. Junhua Wang, Soon-Kyeong Kim, and David A. Carrington. Verifying metamodel coverage of model transformations. In *ASWEC*, pages 270–282. IEEE Computer Society, 2006.

WKK$^+$09. M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schönböck, and W. Schwinger. Right or wrong? - verification of model transformations using colored petri nets. In *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM'09)*, 2009.

Complementary References

Abr96. J.-R. Abrial. *The B book: Assigning Programs to Meanings*. Cambridge University Press, New York, NY, USA, 1996.

Ake09. D. Akehurst. OCL4Java library Canterbury University, 2009. URL: www.cs.ukc.ac.uk/.

AKL03. Aditya Agrawal, Gabor Karsai, and Ákos Lédeczi. An end-to-end domain-driven software development framework. In Ron Crocker and Guy L. Steele Jr., editors, *OOPSLA Companion*, pages 8–15. ACM, 2003.

ATL05. ATLAS. *KM3: Kernel MetaMetaModel*. LINA & INRIA, Manual v0.3 edition, 2005.

ATL06. ATLAS. *ATL: Atlas Transformation Language*. LINA & INRIA, User Manual v0.7 edition, 2006.

BC04. Yves Bertot and Pierre Casteran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.

BCCG07. Reda Bendraou, Benoît Combemale, Xavier Crégut, and Marie-Pierre Gervais. Definition of an executable SPEM 2.0. In *APSEC*, pages 390–397. IEEE Computer Society, 2007.

BGL$^+$00. Saddek Bensalem, Vijay Ganesh, Yassine Lakhnech, César Mu noz, Sam Owre, Harald Rueß, John Rushby, Vlad Rusu, Hassen Saïdi, N. Shankar, Eli Singerman, and Ashish Tiwari. An overview of SAL. In C. Michael Holloway, editor, *LFM 2000: Fifth NASA Langley Formal Methods Workshop*, pages 187–196, Hampton, VA, jun 2000. NASA Langley Research Center.

BJM00. Adel Bouhoula, Jean-Pierre Jouannaud, and José Meseguer. Specification and proof in membership equational logic. *Theor. Comput. Sci.*, 236(1-2):35–132, 2000.

BNL05. Dirk Beyer, Andreas Noack, and Claus Lewerentz. Efficient relational calculation for software analysis. *IEEE Transactions on Software Engineering*, 31(2):2005, 2005.

BRS$^+$00. Michael Balser, Wolfgang Reif, Gerhard Schellhorn, Kurt Stenzel, and Andreas Thums. Formal system development with KIV. In *Fundamental Approaches to Software Engineering*, volume 1783 of *Lecture Notes in Computer Science*, pages 363–366. Springer, 2000.

BRST05. Jean Bézivin, Bernhard Rumpe, Andy Schürr, and Laurence Tratt. Model transformations in practice workshop. In Bruel [Bru06], pages 120–127.

Bru06. Jean-Michel Bruel, editor. *Satellite Events at the MoDELS 2005 Conference, Jamaica. Revised Selected Papers*, volume 3844 of *Lecture Notes in Computer Science*. Springer, 2006.

CA05. Krzysztof Czarnecki and Michal Antkiewicz. Mapping features to models: A template approach based on superimposed variants. In Robert Glück and Michael R. Lowry, editors, *GPCE*, volume 3676 of *Lecture Notes in Computer Science*, pages 422–437. Springer, 2005.

CCR07. Jordi Cabot, Robert Clarisó, and Daniel Riera. UMLtoCSP: a tool for the formal verification of UML/OCL models using constraint programming. In R. E. Kurt Stirewalt, Alexander Egyed, and Bernd Fischer, editors, *ASE*, pages 547–548. ACM, 2007.

CH06. Krzysztof Czarnecki and Simon Helsen. Feature-based survey of model transformation approaches. *IBM Syst. J.*, 45(3):621–645, 2006.

CKTW08. María Victoria Cengarle, Alexander Knapp, Andrzej Tarlecki, and Martin Wirsing. A heterogeneous approach to UML semantics. In Pierpaolo Degano, Rocco De Nicola, and José Meseguer, editors, *Concurrency, Graphs and Models*, volume 5065 of *Lecture Notes in Computer Science*, pages 383–402. Springer, 2008.

Com05. Compuware. OptimalJ 4.0, user's guide, 2005. URL: www.compuware.com/products/optimalj.

CP90. Thierry Coquand and C. Paulin. Inductively defined types. In *COLOG-88: Proceedings of the international conference on Computer logic*, pages 50–66. Springer, 1990.

Dir02. Ravi Dirckze. Java Metadata Interface (JMI) specification. Technical Report JSR 040, Java Community Process, 2002.

FSM$^+$03. F.Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. Grose. *Eclipse Modeling Framework*. Addison-Wesley Professional, 2003.

GB83. Joseph A. Goguen and Rod M. Burstall. Introducing institutions. In Edmund M. Clarke and Dexter Kozen, editors, *Logic of Programs*, volume 164 of *Lecture Notes in Computer Science*, pages 221–256. Springer, 1983.

GJSB05. James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *Java(TM) Language Specification, The (3rd Edition)*. Addison-Wesley Professional, 2005.

Hoa78. C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, August 1978.

IEE90. IEEE. IEEE standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, page 1, 1990. URL: `standards.ieee.org/findstds/standard/610.12-1990.html`.

JK05. Frédéric Jouault and Ivan Kurtev. Transforming models with ATL. In Bruel [Bru06], pages 128–138.

JK09. Kurt Jensen and Lars Michael Kristensen. *Coloured Petri Nets - Modelling and Validation of Concurrent Systems*. Springer, 2009.

JSS01. Daniel Jackson, Ilya Shlyakhter, and Manu Sridharan. A micromodularity mechanism. *SIGSOFT Softw. Eng. Notes*, 26(5):62–73, September 2001.

Kay05. Michael Kay. XSL transformations (XSLT) version 2.0. W3C Working Draft, April 2005. URL: `www.w3.org/TR/xslt20/`.

KBA02. I. Kurtev, J. Bezivin, and M. Aksit. Technological spaces: An initial appraisal. In *International Symposium on Distributed Objects and Applications, DOA 2002*, 2002.

Ken02. Stuart Kent. Model-driven engineering. In Michael Butler, Luigia Petre, and Kaisa Sere, editors, *IFM*, volume 2335 of *Lecture Notes in Computer Science*, pages 286–298. Springer, 2002.

KWB03. Anneke G. Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

Lam06. Leslie Lamport. Checking a multithreaded algorithm with $^+$cal. In Shlomi Dolev, editor, *DISC*, volume 4167 of *Lecture Notes in Computer Science*, pages 151–163. Springer, 2006.

Lan09. Kevin Lano. A compositional semantics of UML-RSDS. *Software and System Modeling*, 8(1):85–116, 2009.

LLC04. Tihamér Levendovszky, László Lengyel, and Hassan Charaf. Implementing a metamodel-based model transformation system. *Buletinul Stiintific al Universitatii "Politehnica" din Timisoara, Romania*, 49(63):89–95, 2004.

LS05. Michael Lawley and Jim Steel. Practical declarative model transformation with Tefkat. In Jean-Michel Bruel, editor, *MoDELS Satellite Events*, volume 3844 of *Lecture Notes in Computer Science*, pages 139–150. Springer, 2005.

LVV$^+$09. Horacio Lopez, Fernando Varesi, Marcelo Viniolo, Daniel Calegari, and Carlos Luna. Estado del arte de lenguajes y herramientas de transformacion de modelos. Technical Report 09-19, InCo-PEDECIBA, 2009.

MCG04. Tom Mens, Krzysztof Czarnecki, and Pieter Van Gorp. 04101 discussion - a taxonomy of model transformations. In Jean Bézivin and Reiko Heckel, editors, *Language Engineering for Model-Driven Software Development*, volume 04101 of *Dagstuhl*

Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2004.

MCM00. P. Lincoln M. Clavel, S. Eker and J. Meseguer. Principles of Maude. In J. Meseguer, editor, *Electronic Notes in Theoretical Computer Science*, volume 4. Elsevier Science Publishers, 2000.

Men10. Tom Mens. Model transformation: A survey of the state-of-the-art. In Sebastien Gerard, Jean-Philippe Babau, and Joel Champeau, editors, *Model Driven Engineering for Distributed Real-Time Embedded Systems*. Wiley - ISTE, 2010.

MFJ05. Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. Weaving executability into object-oriented meta-languages. In Lionel C. Briand and Clay Williams, editors, *MoDELS*, volume 3713 of *Lecture Notes in Computer Science*, pages 264–278. Springer, 2005.

MML07. Till Mossakowski, Christian Maeder, and Klaus Lüttich. The heterogeneous tool set, hets. In Orna Grumberg and Michael Huth, editors, *TACAS*, volume 4424 of *Lecture Notes in Computer Science*, pages 519–522. Springer, 2007.

Mos05. Till Mossakowski. Heterogeneous specification and the heterogeneous tool set. Technical report, Universitaet Bremen, 2005. Habilitation thesis.

NPW02. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

OMG03. OMG. Meta Object Facility (MOF) 2.0 Core Specification. Specification Version 2.0, Object Management Group, 2003.

OMG05. OMG. Unified Modeling Language: Superstructure. Specification Version 2.0, Object Management Group, August 2005.

OMG09. OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation. Final Adopted Specification Version 1.1, Object Management Group, 2009.

OMG10. OMG. Object Constraint Language, version 2.2. Formal Specification formal/2010-02-01, Object Management Group, 2010.

OMG11. OMG. OMG MOF 2 XMI Mapping Specification. Specification Version 2.4.1, Object Management Group, 2011.

Pet94. Mikael Pettersson. RML - a new language and implementation for natural semantics. In Manuel V. Hermenegildo and Jaan Penjam, editors, *PLILP*, volume 844 of *Lecture Notes in Computer Science*, pages 117–131. Springer, 1994.

PR08. Carl Adam Petri and Wolfgang Reisig. Petri net. *Scholarpedia*, 3(4):6477, 2008. URL: http://www.scholarpedia.org/article/Petri_net.

Sch94. Andy Schürr. Specification of graph translators with triple graph grammars. In Ernst W. Mayr, Gunther Schmidt, and Gottfried Tinhofer, editors, *WG*, volume 903 of *Lecture Notes in Computer Science*, pages 151–163. Springer, 1994.

Sch06. Douglas Schmidt. Guest editor's introduction: Model-Driven Engineering. *IEEE Computer*, 39(2):25–31, 2006.

Sel01. Bran Selic. Using UML to model complex real-time architectures. In Peter P. Hofmann and Andy Schürr, editors, *OMER*, volume 5 of *LNI*, pages 16–21. GI, 2001.

SNEC08. Mehrdad Sabetzadeh, Shiva Nejati, Steve Easterbrook, and Marsha Chechik. Global consistency checking of distributed models with TReMer. In *In 30th International Conference on Software Engineering (ICSE'08), 2008. Formal Research Demonstration (To Appear*, 2008.

Tae03. Gabriele Taentzer. AGG: A graph transformation environment for modeling and validation of software. In John L. Pfaltz, Manfred Nagl, and Boris Böhlen, editors, *AGTIVE*, volume 3062 of *Lecture Notes in Computer Science*, pages 446–453. Springer, 2003.

WKS$^+$09. Manuel Wimmer, Gerti Kappel, Johannes Schönböck, Angelika Kusel, Werner Retschitzegger, and Wieland Schwinger. TROPIC: A framework for model transformations on petri nets in color. In Shail Arora and Gary T. Leavens, editors, *OOPSLA Companion*, pages 783–784. ACM, 2009.

WSTL10. Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. SWI-Prolog. *CoRR*, abs/1011.5332, 2010.

Acronyms & Tools

+CAL +CAL Tool [Lam06]. 26

AGG Attributed Graph Grammar Tool [Tae03]. 5, 23, 25–28
Alloy Alloy Language and Tools [JSS01]. 22, 26
API Application Programming Interface. 5, 30
ATL Atlas Transformation Language [JK05]. 3, 5, 14, 24, 28, 30

B B formal Method[Abr96]. 15, 27

CIC Calculus of Inductive Constructions [CP90]. 16, 24
Coq Coq Proof Assistant [BC04]. 15, 29
CPN Colored Petri Net [JK09]. 18, 30
CrocoPat CrocoPat Tool [BNL05]. 23, 25
CSP Communicating Sequential Processes [Hoa78]. 25

DSLTrans DSLTrans [BLA+10]. 23

EMF Eclipse Modeling Framework [FSM+03]. 3, 27

FOL First-Order Logic. 15, 16, 21

GReAT Graph Rewriting and Transformation [AKL03]. 5, 28

Hets Heterogeneous Tool Set (Hets) [Mos05,MML07]. 21
HOL Higher-Order Logic. 21

Isabelle/HOL Isabelle/HOL Tool [NPW02]. 15, 21, 25, 29

Java Java Language [GJSB05]. 5, 22, 29
JMI Java Metadata Interface [Dir02]. 5

Kermeta Kermeta Tool [MFJ05]. 5
KIV KIV Tool [BRS+00]. 29
KM3 Kernel MetaMetaModel (KM3) [ATL05]. 3, 14

Maude Maude System [MCM00]. 20, 21, 23
MDE Model-Driven Engineering [Ken02,Sch06]. 2, 3, 20, 21
MEL Membership Equational Logic [BJM00]. 23
MOF MetaObject Facility [OMG03]. 3, 4, 21, 22
MOMENT MOdel manageMENT Tool [BHM09]. 23

OCL Object Constraint Language [OMG10]. 3, 15–17, 22–24, 26, 28
OCL4Java OCL4Java Library [Ake09]. 22
OptimalJ OptimalJ Tool [Com05]. 5

Petri Net Petri Net [PR08]. 18, 21, 24, 29, 30

QVT Query/View/Transformation [OMG09]. 5, 12, 21–25, 29

RML Relational Meta-Language [Pet94]. 23

SAL Symbolic Analysis Laboratory Tool [BGL+00]. 24, 29
SWI-Prolog SWI-Prolog Tool [WSTL10]. 16, 22, 27

Tefkat Tefkat Tool [LS05]. 5
TGG Triple Graph Grammars [Sch94]. 24–26
TReMer+ TReMer+ Tool [SNEC08]. 25
TROPIC TRansformations On Petri Nets In Color Tool [WKS+09]. 30

UML Unified Modeling Language [OMG05]. 3, 12, 14, 17–19, 24–26, 28
UML-RSDS UML Reactive System Development Support [Lan09]. 27
UML-RT UML-RT Language [Sel01]. 25
UMLtoCSP UMLtoCSP Tool [CCR07]. 24

VIATRA VIsual Automated model TRAnsformations Tool [CHM+02]. 5, 24
VMTS Visual Modeling and Transformation System [LLC04]. 22

XMI XML Metadata Interchange (XMI) [OMG11]. 3, 21
XSLT Extensible Stylesheet Language Transformation (XSLT) [Kay05]. 5
xSPEM eXecutable SPEM 2.0 [BCCG07]. 24