Institution-Based Semantics for MOF and QVT-Relations

Daniel Calegari[1] and Nora Szasz[2]

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

Abstract. To cope with formal verification issues within the Model-Driven Engineering (MDE) paradigm, a separation of duties between software developers is usually proposed: MDE experts define models and transformations, while formal verification experts conduct the verification process. This is often aided by (semi)automatic translations form the MDE elements to their formal representation in the semantic domain used for verification. From a formal perspective, this requires semantic-preserving translations between the MDE elements and the semantic domain. The aim of this paper is to present formal semantics for the MOF and QVT-Relations languages which are standard languages for defining metamodels and model transformations, respectively. The semantics is based on the Theory of Institutions and reflect the conformance relation between models and metamodels, and the satisfaction of transformation rules between pairs of models. The theory assists in the definition of semantic-preserving translations between our institutions and other logics which will be used for verification.

Keywords: MOF, QVT-Relations, formal semantics, Theory of Institutions, verification.

1 Introduction

The Model-Driven Engineering paradigm (MDE, [1]) envisions a software development life-cycle driven by 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. The Object Management Group (OMG) has conducted a standardization process of languages for MDE. They defined the MetaObject Facility (MOF, [2]) as the language for metamodeling as well as three transformation languages with different transformation approaches. The Query/View/Transformation Relations (QVT-Relations, [3]) is one of those languages and follows a relational approach which consists of defining transformation rules as mathematical relations between source and target elements. Since the quality of the whole development

J. Iyoda and L. de Moura (Eds.): SBMF 2013, LNCS 8195, pp. 34–50, 2013.

process strongly depends on the quality of the models and model transformations, verification is a must, and in some cases formal methods arise as a tool for strengthening verification results. To cope with this situation, a separation of duties between software developers is usually proposed. 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 [4], 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, while 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 for verification purposes.

We are exploring a comprehensive formal environment enabling this scheme. This environment requires semantic-preserving translations between the MDE elements and the chosen semantic domain. Moreover, different logics (e.g. modal logic, predicate logic) can be used by verification experts. In this context, the biggest problem is perhaps the maintenance of multiple formal representations of the same MDE elements and the complexity of linking different semantic domains to perform a comprehensive verification using multiple semantic domains.

The aim of this paper is to present formal semantics for the MOF and the QVT-Relations languages in a flexible way to solve the problems described before. We base our proposal on the heterogeneous specification approach [5,6], which consists in having different mathematical formalism for expressing different parts of the overall problem and defining semantic-preserving mappings in order to allow "communication" between the formalisms. This approach uses as a basis the Theory of Institutions [7]. Using this theory we define institutions to represent the conformance relation between MOF models and metamodels and the satisfaction of QVT-Relations transformation rules between pairs of models. The theory also assists in the definition of semantic-preserving translations between our institutions and other logics which will be used for verification.

The remainder of the paper is structured as follows. In Section 2 we introduce the elements involved in the MDE technical space which will be part of this work and we introduce a running example. Then, in Section 3 we summarize the general schema we follow for defining formal semantics based on the Theory of Institutions. In Section 4 we formally define an institution for MOF, and in Section 5 we define the institution for QVT-Relations. Finally, in Section 6 we present some conclusions and guidelines for future work.

2 An Introduction to the MDE Technical Space

In MDE everything is a model, i.e. 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. MOF is a standard language for metamodeling. A metamodel defines classes which can belong to a hierarchical structure and some of them must be defined as abstract (there are no instances of them). Any class has properties which can be attributes (named elements

with an associated type which can be a primitive type or another class) and associations (relations between classes in which each class plays a role within the relation). Every property has a multiplicity which constraints the number of elements that can be related through the property. 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, [8]).

Let us consider a simplified version of the well-known Class to Relational model transformation [3]. The metamodel on the left-hand side of Figure 1 defines UML class diagrams, where classifiers (classes and primitive types as string, boolean, integer, etc.) are contained in packages. Classes can contain one or more attributes and may be declared as persistent, whilst attributes have a type that is a primitive type. On the right-hand side of Figure 1 there is an example of a model composed by a persistent class of name `ID` within a package of name `Package`. The class has an attribute of name `value` and type `String`.

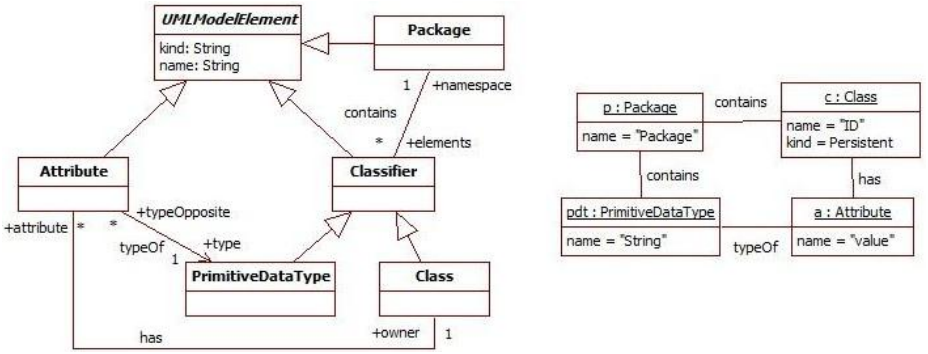

Fig. 1. Class metamodel and model of the example

A model transformation (or just transformation from now on) basically takes as input a model conforming to certain metamodel and produces as output another model conforming to another metamodel (possibly the same). QVT-Relations follows a relational approach which consists on defining transformation rules as mathematical relations between source and target elements. Although transformations can be defined between multiple metamodels at the same time, we will only consider a source and a target metamodel.

A transformation can be viewed as a set of interconnected relations which are of two kinds: top-level relations which must hold in any transformation execution, and non-top-level relations which are required to hold only when they are referred from another relation. We can view a relation as having the following abstract structure [3]:

```
[top] relation R {
    <R_var_set> <R_par_set>
    Domain {
        <domain_k_var_set> <domain_k_pat>
    } //k = 1,2
    [when <when_var_set> <when_cond>]
    [where <where_cond>]
}
```

Every relation has a set `<R_var_set>` of variables occurring in the relation, which are particularly used within the domains (`<domain_k_var_set>`) and in the `when` clause (`<when_var_set>`). Each relation defines a source and a target pattern `<domain_k_pat>` which is used to find matching sub-graphs in a model and can be viewed as a graph of typed pattern elements and pattern links, together with a predicate which must hold. Relations can also contain `when` (`<when_cond>`) and `where` (`<where_cond>`) clauses. A `when` clause specifies the conditions under which the relationship needs to hold, whilst the `where` clause specifies the condition that must be satisfied by all model elements participating in the relation. The `when` and `where` clauses, as well as the predicate of a pattern, may contain arbitrary boolean OCL expressions in addition to the relation invocation expressions. Finally, any relation can define a set of primitive domains which are data types used to parameterize the relation (`<R_par_set>`).

The standard checking semantics states that a rule holds if for each valid binding of variables of the `when` clause and variables of domains other than the target domain, that satisfy the `when` condition and source domain patterns and conditions, there must exist a valid binding of the remaining unbound variables of the target domain that satisfies the target domain pattern and `where` condition.

The Class to Relational transformation basically describes how persistent classes within a package are transformed into tables within a schema. Attributes of a class are transformed into columns of the corresponding table, and the primary key is defined by default. Below we show an excerpt of this transformation.

```
transformation umlToRdbms(uml:SimpleUML, rdbms:SimpleRDBMS) {
  top relation PackageToSchema {
    pn: String;
    domain uml p:Package {name=pn};
    domain rdbms s:Schema {name=pn};
  }
  top relation ClassToTable {
    cn, prefix: String;
    domain uml c:Class {namespace=p:Package {},kind='Persistent',name=cn};
    domain rdbms t:Table {schema=s:Schema {}, name=cn,
       column=cl:Column {name=cn+'_tid', type='NUMBER'},
       key=k:Key {name=cn+'_pk', column=cl}};
    when  { PackageToSchema(p,s); }
    where { prefix = ''; AttributeToColumn(c, t, prefix); }
  }
  relation AttributeToColumn { ... }
}
```

3 An Environment for Verification

We are exploring a comprehensive environment for the formal verification of different aspects of a model transformation using heterogeneous verification approaches [9]. The environment is based on representing models (from now on SW-models), metamodels, the conformance relation, transformations and verification properties in some consistent and interdependent way following the heterogeneous specification approach [5,6]. This approach is based on providing *Institutions* for the languages which are part of the environment. The concept of Institution [7] was originally introduced to formalize the notion of logical system, and many different logics as first-order, modal, rewriting, among others have been shown to be institutions. Informally, an institution consists of a collection of signatures (vocabularies for constructing sentences in a logical system), signature morphisms (allowing many different vocabularies at once), a collection of sentences and models (providing semantics) for a given signature, and a satisfaction relation of sentences by models, such that when signatures are changed (by a signature morphism), satisfaction of sentences by models changes consistently. The notion of an institution can be used to represent any specification language since it provides ways of representing the syntax and semantics of the language, as well as the relation between them by means of a satisfaction relation between them, as in [5]. In this work we provide an institution for QVT-Relations check-only unidirectional transformations. This kind of transformations only checks if a target model is the result of transforming the source SW-model according to the transformation rules. This institution needs a representation of SW-models and metamodels, therefore we first define an institution for MOF for expressing the conformance relation between them.

In order to use our institutions for verification purposes, there are two alternatives. The first one is to extend the institutions from a proof-theoretic point of view by defining a *logic*, i.e. equipping the institutions with an entailment system on sentences for conducting formal proofs. The second alternative is to formally translate our institutions into another logic. This can be done through *institution comorphisms* [10], which capture how a *weaker* institution can be represented in a *stronger* and *richer* one. The importance of comorphisms is such that it is possible (in some cases) to re-use (*borrow*) the entailment systems of an institution in order to prove properties.

We take the second alternative and define comorphisms from our institutions to a host logic and supplement this information with properties specified in the host logic. In particular, we are in the process of defining a comorphism to the Common Algebraic Specification Language (CASL, [11]), a general-purpose specification language. The institution underlying CASL is the sub-sorted partial first-order logic with equality and constraints on sets $SubPCFOL^{=}$, a combination of first-order logic and induction with subsorts and partial functions. The importance of CASL is that it is the main language within the Heterogeneous Tool Set (Hets, [6]), which is a tool meant to support heterogeneous multi-logic specifications. Hets allows defining institutions and comorphisms, and also 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. Hets already supports several interconnected logics (e.g. first-order and modal logics, among others). To the best of our knowledge, Hets does not support the MDE paradigm, i.e. it does not have specific languages for the specification of MDE elements. We plan to include our institutions as logics in Hets, in such a way that a developer can import a transformation (which is automatically translated into CASL through the comorphism), use the logics within Hets to specify additional verification properties which must be addressed, and perform the verification assisted by the tool.

3.1 Defining the Institutions

In Section 4 we define the institution for the MOF-based conformance relation, basing our proposal on the institution defined for UML class diagrams in [12,13]. Unlike [12], in our definition there are no derived relations (not used in transformations), the signature has an explicit representation of abstract classes and datatypes, and there are only 2-ary properties (associations and attributes). We also use an explicit syntactic representation of SW-models within the signature. In [13], instances (class objects and type values) are represented within the signature. However, there is no representation of links between these elements since they are used for other purposes. Moreover, unlike MOF, we do not consider aggregation, uniqueness and ordering properties within a property end, operations on classes, or packages. Properties and operations are not commonly used within transformations, whereas packages are just used for organizing metamodel elements. We follow the schema in Figure 2. From any metamodel we can derive a signature with a representation of types, properties, and SW-models, and a set of formulas stating invariants which must hold on every conforming SW-model. Up to now we have considered multiplicity constraints. However, it will be possible to add other kind of constraints through comorphisms as explained before. Any institution model (from now on just model) is a semantic representation of a potentially conforming SW-model. The model is composed by objects and relations between them, which must satisfy the multiplicity constraints. This allows us to define the satisfaction relation answering the question: does the SW-model conform to the metamodel?

In Section 5 we also define an institution for QVT-Relations check-only unidirectional transformations. For the definition of this institution we follow the schema shown in Figure 3. For the definition we do not consider black-box operations or rule and transformation overriding since they are advanced features not commonly used in practice. We neither consider keys definition since they are not used within the checking semantics. The institution takes the institutional representation of the source and target elements and supplements the formulas with a representation of the transformation rules. In this case, the satisfaction relation also answers the question: is the target SW-model the result of transforming the source SW-model according to the transformation rules?

As we mentioned before, the `when` and `where` clauses, as well as the predicate of a pattern, may contain arbitrary boolean OCL expressions. From a formal

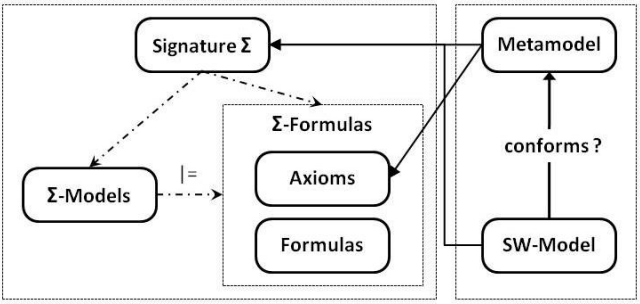

Fig. 2. The conformance relation as an institution

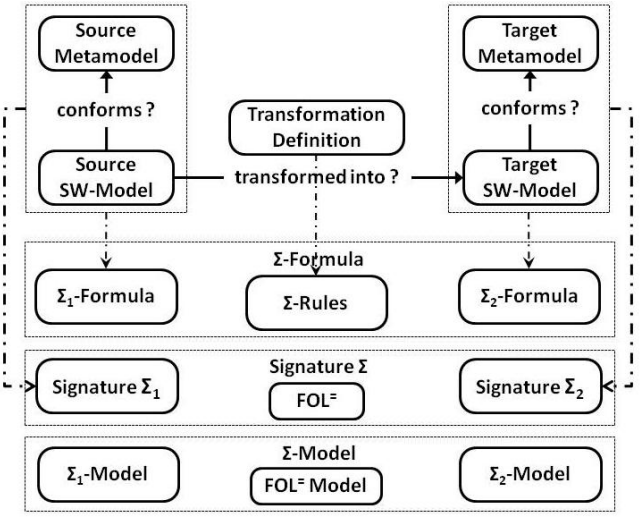

Fig. 3. A model transformation as an institution

perspective we would rather have an institution for OCL which would allow us to use the language not only for constraining the transformation rules, but also for expressing general constraints on metamodels. Unfortunately there is no institution for OCL, which is left for future work. However, in our work we consider an institution for first-order logic with equality ($FOL^{=}$) as defined in [14]. With this decision we are not losing expressive power (there are works [15] with the aim of expressing OCL into first-order logic).

3.2 Related Work

There are many works defining the semantics of MOF and the conformance relation in terms of a shallow embedding of the language by providing a syntactic

translation into another one, e.g. into first-order logic [16] and rewriting logic [17]. We, on the contrary, do not want to depend on a general logic but to define a generic and minimal infrastructure allowing translations to other logics as needed. There are also some works with an algebraic/institutional approach, e.g. [18,19]. In these works the authors propose two alternatives: a generic algebraic representation of metamodels without an explicit representation of models, and concrete institutions for each metamodel. Unlike these works, we avoid the burden of defining a new institution for each metamodel, explicitly representing models to be used within proofs. In [12,13] the authors define institutions for simple and stereotyped UML Class Diagrams. As we said, we adapt those works for the purpose of defining the institution for the conformance relation. Finally, in [20] the authors define the semantics of class diagrams with OCL constraints by defining a translation into CASL. Although this is also a shallow embedding, as explained before, the translation (and the one proposed in [13]) could be useful for defining the comorphism from our institution to CASL.

With respect to QVT-Relations, there are also works defining the semantics of QVT-Relations in terms of a shallow embedding of the language, e.g. into rewriting logic [21] and coloured petri nets [22]. There are also embeddings into specific tools, as in the case of Alloy [23] and KIV [24], which provide model checking capabilities. As said before, we do not follow this approach. Moreover, in [18] transformations are represented as institution comorphisms, which is somehow restrictive since it assumes a semantic relation between metamodels. Finally, in [25] the authors present a formal semantics for the QVT-Relations check-only scenario based on algebraic specification and category theory. The definition of the institution is much more complex than ours and the work does not envision a scenario in which the elements of the transformation are translated to other logics for verification.

4 An Institution for MOF

In this section we present the formal definition of an institution $\mathcal{I}^{\mathrm{C}}$ for the MOF-based conformance relation. As said before, this definition is based on the institutions for UML class diagrams defined in [12,13], but adapted for representing the conformance relation. Along the definition we will illustrate the concepts introduced with the example presented in Section 2.

A class hierarchy is a partial order $\boldsymbol{C} = (C, \leq_C)$ where C is a set of class names, and $\leq_C \subseteq C \times C$ is the subclass (inheritance) relation. By $\boldsymbol{T}(\boldsymbol{C})$ we denote the *type extension* of $\boldsymbol{C}$ by primitive types and type constructors. $\boldsymbol{T}(\boldsymbol{C})$ is likewise a class hierarchy $(T(C), \leq_{T(C)})$ with $C \subseteq T(C)$ and $\leq_C \subseteq \leq_{T(C)}$. As in [13], in order to provide generic access to primitive types, like Boolean, and String, we treat these as built-ins with a standard meaning (they must be defined within $T(C)$). All other classes are assumed to be inhabited, i.e., to contain at least one object. However, unlike [13] in which it is assumed the existence of an object *null*, we impose that if $c \in C|_{abstract}$ then there exists another $c' \in T(C)$ downwards in the hierarchy having at least one object.

As we mentioned before, from a metamodel we can derive a signature $\Sigma = (\boldsymbol{T}, \boldsymbol{P}, \boldsymbol{M})$ declaring:

- A type extension of a finite class hierarchy $\boldsymbol{T} = (T(C), \leq_{T(C)}, C\,|_{abstract})$ extended with a subset $C|_{abstract} \subseteq C$ denoting abstract classes.
- A properties declaration (attributes and associations) $\boldsymbol{P} = (R, P)$ where R is a finite set of role names and P is a finite set $(p_w)_{w \in (R \times T(C)) \times (R \times T(C))}$ of property names indexed over pairs of a role name and a class (or type) name, such that for any class or type name $c \in C$, the role names of the properties in which any $c' \leq_{T(C)} c$ is involved are all different. If $p_w \in P$ with $w = ((r_1, c_1)(r_2, c_2))$, we write $p(r_1 : c_1, r_2 : c_2) \in P$.
- A SW-model declaration (instances and links) $\boldsymbol{M} = (I, L)$ where I is a finite set of instances of the form $o : c$ with $c \in T(C)$; and L is a finite set of links between instances of the form $p_w(x, y)$ with $p_w \in P$, $w = ((r_1, c)(r_2, d))$, $x : c,\ y : d\ \in I$.

From a metamodel it is also possible to derive a set of formulas (multiplicity constraints) constraining the set of SW-models conforming to it. Given a signature as defined before, any Σ-formula is defined by:

$$\begin{aligned} \Phi &::= \#\Pi = n \mid n \leq \#\Pi \mid \#\Pi \leq n \\ \Pi &::= R \bullet P \end{aligned}$$

where $n \in \mathbb{N}$. The #-expressions return the number of links in a property when some roles are fixed. We use $\bullet$ as the select/partition operator in Π representing the selection of the elements in the opposite side of role R in property P.

Let $\Sigma_i = (\boldsymbol{T}_i, \boldsymbol{P}_i, \boldsymbol{M}_i)$ $(i = 1, 2)$ with $\boldsymbol{T}_i = (T(C_i), \leq_{T(C_i)}, C_i\,|_{abstract})$, $\boldsymbol{P}_i = (R_i, P_i)$, and $\boldsymbol{M}_i = (I_i, L_i)$. A signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ is a tuple of maps $\langle \sigma_T, \sigma_R, \sigma_P, \sigma_I \rangle$ between class names, role names, property names, and instances. Signature morphisms extend to formulas over Σ_1 as follows. Given a Σ_1-formula φ, $\sigma(\varphi)$ is the canonical application of the signature morphism to every role and property in the formula such that $\sigma(r \bullet p) = \sigma_R(r) \bullet \sigma_P(p)$.

Given a class hierarchy $\boldsymbol{C} = (C, \leq_C)$, a $\boldsymbol{C}$-object domain $\boldsymbol{O}$ is a family $(O_c)_{c \in C}$ of sets of object identifiers verifying $O_{c_1} \subseteq O_{c_2}$ if $c_1 \leq_C c_2$. Given moreover a type extension $\boldsymbol{T}$, the *value extension* of a $\boldsymbol{C}$-object domain $\boldsymbol{O} = (O_c)_{c \in C}$ by primitive values and value constructions, which is denoted by $\boldsymbol{V}^{\boldsymbol{T}}_{\boldsymbol{C}}(\boldsymbol{O})$, is a $\boldsymbol{T}(\boldsymbol{C})$-object domain $(V_c)_{c \in T(C)}$ such that $V_c = O_c$ for all $c \in C$. We consider disjoint sets of objects within the same hierarchical level.

We adapt the definition of a Σ-interpretation in order to 'reduce' the interpretation to those elements and relations in $\boldsymbol{M}$, i.e. there is an isomorphism between these elements and those in the interpretation. A Σ-interpretation $\mathcal{I}$ consists of a tuple $(\boldsymbol{V}^{\boldsymbol{T}}_{\boldsymbol{C}}(\boldsymbol{O}), \boldsymbol{A}, K^{\mathcal{I}})$ where

- $\boldsymbol{V}^{\boldsymbol{T}}_{\boldsymbol{C}}(\boldsymbol{O}) = (V_c)_{c \in T(C)}$ is a $\boldsymbol{T}(\boldsymbol{C})$-object domain
- $\boldsymbol{A}$ contains a relation $p^{\mathcal{I}} \subseteq V_{c_1} \times V_{c_2}$ for each relation name $p(r_1 : c_1, r_2 : c_2) \in P$ with $c_1, c_2 \in T(C)$
- $K^{\mathcal{I}}$ maps each $o : c \in I$ to an element of V_c
- $c_2 \in C|_{abstract}$ implies $O_{c_2} = \bigcup_{c_1 \leq_C c_2} O_{c_1}$

- $K^{\mathcal{I}}(o_1 : c) \neq K^{\mathcal{I}}(o_2 : d)$ iff $o_1 : c \neq o_2 : d$
- $V_c = \bigcup_c K^{\mathcal{I}}(o : c)$ with $o : c \in I$, for all $c \in T(C)$
- $p^{\mathcal{I}} = \{(K^{\mathcal{I}}(x : c), K^{\mathcal{I}}(y : d)) \mid p_w(x, y) \in L, x : c,\ y : d\ \in I\}$

Given a Σ-interpretation $\mathcal{I} = (\boldsymbol{V_C^T}(\boldsymbol{O}), \boldsymbol{A}, K^{\mathcal{I}})$, the interpretation evaluates relations as follows: if $p(r_1 : c_1, r_2 : c_2)$ then $(r_i \bullet p)^{\mathcal{I}} = \{\{t \in p^{\mathcal{I}} \mid \pi_i(t) = o\} \mid o \in V_{c_i}\}$ $(i = 1, 2)$. The evaluation $(r_i \bullet p)^{\mathcal{I}}$ gives a set of sets of pairs of semantic elements connected through property p, grouped by the semantic elements having role r_i. Note that this set can be empty if the element with role r_i is not connected with any one.

Given a signature Σ, a formula φ, and a Σ-interpretation $\mathcal{I}$, the interpretation satisfies φ, written $\mathcal{I} \models_\Sigma \varphi$, if one of the following conditions holds:

- φ is $\#(r \bullet p) = n$ and $|S| = n$ for all $S \in (r \bullet p)^{\mathcal{I}}$
- φ is $n \leq \#(r \bullet p)$ and $n \leq |S|$ for all $S \in (r \bullet p)^{\mathcal{I}}$
- φ is $\#(r \bullet p) \leq n$ and $|S| \leq n$ for all $S \in (r \bullet p)^{\mathcal{I}}$

This means that the number of elements related through a property p with any element with role r in such property, satisfies the multiplicity constraints. This definition can be trivially defined for a set of formulas Φ.

Finally, the satisfaction condition holds for given signatures Σ_i $(i = 1, 2)$, a signature morphism $\sigma : \Sigma_1 \to \Sigma_2$, a Σ_2-interpretation $\mathcal{I}$, and a Σ_1-formula ψ: $\mathcal{I}|_\sigma, \models_{\Sigma_1} \psi$ iff $\mathcal{I} \models_{\Sigma_2} \sigma(\psi)$. This can be trivially extended to a set of formulas.

Given that the satisfaction condition holds we can state that $\mathcal{I}^{\mathrm{C}}$ consisting of signatures, morphisms, formulas, interpretations, reducts, and the satisfaction relation, defines an institution. For space reasons we omit here several definition (e.g. signature morphisms, reducts) and proofs, which can be completely found in [26].

4.1 Running Example

From the class metamodel and the SW-model in Figure 1 we derive the signature $(\boldsymbol{T}, \boldsymbol{P}, \boldsymbol{M})$ with $\boldsymbol{T} = (T(C), \leq_{T(C)}, C|_{abstract})$, $\boldsymbol{P} = (R, P)$, and $\boldsymbol{M} = (I, L)$ such that:

$$
\begin{aligned}
T(C) &= \{\text{UMLModelElement}, \text{Package}, ..., \text{String}\} \\
\leq_{T(C)} &= \{\text{Package} \leq_{T(C)} \text{UMLModelElement}, ...\} \\
C|_{abstract} &= \{\text{UMLModelElement}\} \\
R &= \{\text{namespace}, \text{elements}, \text{type}, \text{typeOpposite}, ...\} \\
P &= \{\text{contains}(\text{namespace} : \text{Package}, \text{elements} : \text{Classifier}), \\
&\quad \text{name}(\text{UMLModelElement} : \text{UMLModelElement}, \text{name} : \text{String}), \\
&\quad \text{typeOf}(\text{typeOpposite} : \text{Attribute}, \text{type} : \text{PrimitiveDataType}), ...\} \\
I &= \{\text{p} : \text{Package}, \text{c} : \text{Class}, \text{a} : \text{Attribute}, ..., \text{String} : \text{String}\} \\
L &= \{\text{contains}(\text{p}, \text{c}), \text{contains}(\text{p}, \text{pdt}), \text{has}(\text{c}, \text{a}),\ \text{type}(\text{a}, \text{pdt}), \\
&\quad \text{name}(\text{p}, \text{Package}),\ \text{kind}(\text{c}, \text{Persistent}),\ \text{name}(\text{pdt}, \text{String}), ...\}
\end{aligned}
$$

The set of formulas φ is defined by:

$$\varphi = \{\#(\text{UMLModelElement} \bullet \text{name}) = 1,\ \#(\text{UMLModelElement} \bullet \text{kind}) = 1,\\ \#(\text{elements} \bullet \text{contains}) = 1,\ \#(\text{attribute} \bullet \text{has}) = 1, ...\}$$

An interpretation $\mathcal{I}$ can be defined as follows, in which each element has a correspondence with one in the signature:

- A $\boldsymbol{T}(\boldsymbol{C})$-object domain consisting of
  $V_{\text{Class}} = \{c1\}$
  $V_{\text{PrimitiveDataType}} = \{pdt1\}$
  $V_{\text{Package}} = \{p1\}$
  $V_{\text{Attribute}} = \{a1\}$
  $V_{\text{String}} = \{Pac, Str, Per, nul, ID, val\}$

- A set $\boldsymbol{A}$ consisting of relations
  $\text{contains}^{\mathcal{I}} = \{(p1, c1), (p1, pdt1)\}$
  $\text{name}^{\mathcal{I}} = \{(p1, Pac), (c1, ID), (c2, nul), (a1, val)\}$
  $\text{kind}^{\mathcal{I}} = \{(p1, nul), (c1, Per), (a1, nul), (pdt1, nul)\}$
  $\text{type}^{\mathcal{I}} = \{(a1, pdt1)\}$
  ...

The property contains(namespace : Package, elements : Classifier) represents that a package contains classifiers. The interpretation $\mathcal{I}$ has the following interpretation of this property: $\text{contains}^{\mathcal{I}} = \{(p1, c1), (p1, c2)(p1, pdt1), (p1, pdt2)\}$, such that there is only one package object $p1$, and it contains two classes ($c1$ and $c2$) and two primitive datatype objects ($pdt1$ and $pdt2$). This interpretation evaluates $(\text{namespace} \bullet \text{contains})^{\mathcal{I}}$ as the set $\{(p1, c1), (p1, c2), (p1, pdt1), (p1, pdt2)\}$ since there is only one object with role namespace which is the package object $p1$, and those elements in the opposite side of the property are those in $\text{contains}^{\mathcal{I}}$.

Now, we check that $\mathcal{I}, \beta \models_{\Sigma} \varphi$ for every formula φ defined before. For example it holds in the following cases.

- $\#(\text{UMLModelElement} \bullet \text{name}) = 1$ and $|S| = 1$
  for all $S \in (\text{UMLModelElement} \bullet \text{name})^{\mathcal{I}} =$
  $\{\{(p1, Pac)\}, \{(c1, ID)\}, \{(a1, val)\}, \{(pdt1, Str)\}\}$

- $\#(\text{elements} \bullet \text{contains}) = 1$ and $|S| = 1$
  for all $S \in (\text{elements} \bullet \text{contains})^{\mathcal{I}} = \{\{(p1, c1)\}, \{(p1, pdt1)\}\}$

5 An Institution for QVT-Relations

We finally introduce an institution $\mathcal{I}^{\text{QVT}}$ for QVT-Relations check-only unidirectional transformations, and then we continue illustrating the concepts introduced with the example presented in Section 2.

A signature in $\mathcal{I}^{\text{QVT}}$ is a triple $\langle \Sigma_1^{\text{C}}, \Sigma_2^{\text{C}}, \Sigma^{\text{FOL}} \rangle$ with $\mathcal{I}^{\text{C}}$-signatures Σ_i^{C} $(i = 1, 2)$ representing the source and target metamodels and models of

the transformation, and a $FOL^{=}$ signature Σ^{FOL} such that there are sorts for every type ($\bigcup_i T(C_i) \subseteq S$) and there is a predicate for each property declaration ($\bigcup_i P_i \subseteq \Pi$). We assume that there are no name clashes (types, roles and properties) between source and target metamodels. In fact, if a transformation has the same source and target metamodels, we can use a prefix to identify elements on each side. A signature morphism is defined as a triple of morphisms of the corresponding institutions.

A Σ-formula is of the form $\langle \varphi_1^{\mathrm{C}}, \varphi_2^{\mathrm{C}}, \varphi^{\mathrm{rules}} \rangle$ such that φ_i^{C} is a Σ_i^{C}-formula and φ^{rules} is a formula representing the transformation specification. i.e. a tuple $\langle \mathrm{Rules}, \mathrm{top} \rangle$ such that Rules is the set of transformation rules, and top $\subseteq$ Rules the set of top rules of the transformation.

A rule Rule $\in$ Rules is a tuple $\langle \mathrm{VarSet}, \mathrm{Pattern}_i\ (i = 1, 2), \mathrm{when}, \mathrm{where} \rangle$ such that VarSet $\subseteq X^s$ with $s \in S$ is the set of variables of the rule, $\mathrm{Pattern}_i$ $(i = 1, 2)$ are the source and target patterns, and when/where are the `when`/`where` clauses of the rule, respectively. We will denote by k_VarSet (k = 1, 2) the variables used in pattern k that do neither occur in the other domain nor in the `when` clause.

A pattern $\mathrm{Pattern}_i$ $(i = 1, 2)$ is a tuple $\langle E_i, A_i, Pr_i \rangle$ such that $E_i \subseteq (X^c)_{c \in C_i}$ is a set of class-indexed variables, A_i is a set of elements representing associations of the form $rel(p, x, y)$ with $p \in P_i$ and $x, y \in E_i$, and Pr_i is a $FOL^{=}$-formula.

A `when` clause is a pair $\langle \mathrm{when}_c, \mathrm{when}_r \rangle$ such that when_c is a $FOL^{=}$-formula with variables in VarSet, and when_r is a set of pairs of transformation rules and set of variables which are the parameters used for the invocation of each rule. We will denote by WhenVarSet the set of variables ocurring in the `when` clause. Finally, a `where` clause is a pair $\langle \mathrm{where}_c, \mathrm{where}_r \rangle$ such that where_c is a $FOL^{=}$-formula with variables in VarSet, and where_r is a set of pairs of transformation rules and set of variables (parameters). Only variables used in a `where` clause (as `prefix` in the example) are contained in 2_VarSet.

A Σ-model is a triple $\langle \mathcal{M}_1^{\mathrm{C}}, \mathcal{M}_2^{\mathrm{C}}, \mathcal{M}^{\mathrm{FOL}} \rangle$ of $\mathrm{Sign}_i^{\mathrm{C}}$ $(i = 1, 2)$ models, and a $\mathrm{Sign}^{\mathrm{FOL}}$ first-order structure, such that the interpretation of elements in $\mathrm{Sign}_i^{\mathrm{C}}$ must be the same in $\mathcal{M}_i^{\mathrm{C}}$ and $\mathcal{M}^{\mathrm{FOL}}$. This means that $|D|_t = V_t$. $\forall t \in \bigcup_i T(C_i)$, and $p_D = p^{\mathcal{I}}$. $\forall p \in \bigcup_i P_i$. In the case of $t \in T(C) \backslash C$ (primitive types) we have that $V_t \subseteq |D|_t$ since $\mathcal{M}^{\mathrm{FOL}}$ can have more elements than those in the source and target institutions: type constants (e.g. the empty string) and elements created using type constructors (e.g. new strings using type constructor ++).

Given variables $X^s = (X^s)_{s \in S}$, the binding of a variable $x^c \in X^c$, denoted by $|x^c|$, is the set of possible interpretations of such a variable which corresponds to the carrier set of the corresponding sort, i.e. $|x^c| = |D|_c$. Moreover, the binding of a set of variables $(x_1, ..., x_n)$, denoted by $|(x_1, ..., x_n)|$, is defined as $\{(y_1, ..., y_n) \mid y_i \in |x_i|\ (i = 1..n)\}$. We can also view $|(x_1, ..., x_n)|$ as a set of variable assignments. We denote by $\mu[x_1, ..., x_n]$ the function with an assignment for variables $x_1, ..., x_n$. We also denote by $\mu_1 \cup \mu_2$ an assignment unifying the former ones, assuming that if there is variable clash, the assignment takes for those variables the values in μ_2.

A `when` clause $\langle \mathrm{when}_c, \mathrm{when}_r \rangle$ is satisfied with respect to a first-order structure $\mathcal{M}^{\mathrm{FOL}}$ and a variable assignment μ, denoted by $\mathcal{M}^{\mathrm{FOL}}, \mu \models \langle \mathrm{when}_c, \mathrm{when}_r \rangle$ if

$\mathcal{M}^{\text{FOL}}, \mu \models_{\text{FOL}} \text{when}_c \ \wedge \ (\forall (r, v) \in \text{when}_r.\ \mathcal{M}^{\text{FOL}}, \mu[v] \models r)$ Here, $\models_{\text{FOL}}$ is the satisfaction relation in $FOL^{=}$, and $\models$ is the satisfaction of the parametric transformation rule r using the variable assignment $\mu[v]$ as a parameter. The satisfaction of a `where` clause is defined in the same way.

A pattern $\text{Pattern} = \langle E, A, Pr \rangle$ is satisfied with respect to a first-order structure $\mathcal{M}^{\text{FOL}}$ and a variable assignment μ (which must include a valuation for the elements in E), denoted by $\mathcal{M}^{\text{FOL}}, \mu \models \text{Pattern}$ if there is a matching subgraph $\forall\ rel(p, x, y) \in A.\ (p_D(\mu(x), \mu(y)) \in\ \mathcal{M}^{\text{FOL}})$, and the predicate holds in $FOL^{=}$ $(\mathcal{M}^{\text{FOL}}, \mu \models_{\text{FOL}} Pr)$.

A rule $\text{Rule} = \langle \text{VarSet}, \text{Pattern}_i\ (i = 1, 2), \text{when}, \text{where} \rangle$ is satisfied with respect to a first-order structure $\mathcal{M}^{\text{FOL}}$ and a variable assignment μ, denoted by $\mathcal{M}^{\text{FOL}}, \mu \models \text{Rule}$ if one of the following properties hold.

1. If $\text{WhenVarSet} = \emptyset$

$$
\begin{aligned}
&\forall\ \mu^1[x_1, ..., x_n] \in\ |\text{VarSet} \backslash \text{2\_VarSet}|, \\
&\quad (\ \mathcal{M}^{\text{FOL}}, (\mu^1[x_1, ..., x_n] \cup \mu) \models \text{Pattern}_1 \rightarrow \\
&\qquad \exists\ \mu^2[y_1, ..., y_m] \in |\text{2\_VarSet}|, \\
&\qquad\quad (\ \mathcal{M}^{\text{FOL}}, (\mu^1 \cup \mu^2 \cup \mu) \models \text{Pattern}_2 \wedge \\
&\qquad\qquad \mathcal{M}^{\text{FOL}}, (\mu^1 \cup \mu^2 \cup \mu) \models \text{where}))
\end{aligned}
$$

2. If $\text{WhenVarSet} \neq \emptyset$

$$
\begin{aligned}
&\forall\ \mu^w[z_1, ..., z_o] \in\ |\text{WhenVarSet}|, \\
&\quad (\ \mathcal{M}^{\text{FOL}}, (\mu^w[z_1, ..., z_o] \cup \mu) \models \text{when} \rightarrow \\
&\qquad \forall\ \mu^1[x_1, ..., x_n] \in\ |\text{VarSet} \backslash (\text{WhenVarSet} \cup \text{2\_VarSet})|, \\
&\qquad\quad (\ \mathcal{M}^{\text{FOL}}, (\mu^1 \cup \mu^w \cup \mu) \models \text{Pattern}_1 \rightarrow \\
&\qquad\qquad \exists\ \mu^2[y_1, ..., y_m] \in |\text{2\_VarSet}|, \\
&\qquad\qquad\quad (\ \mathcal{M}^{\text{FOL}}, (\mu^1 \cup \mu^2 \cup \mu^w \cup \mu) \models \text{Pattern}_2 \wedge \\
&\qquad\qquad\qquad \mathcal{M}^{\text{FOL}}, (\mu^1 \cup \mu^2 \cup \mu^w \cup \mu) \models \text{where})))
\end{aligned}
$$

The satisfaction relation is defined in such a way that a model $\mathcal{M}$ satisfies φ, written $\mathcal{M} \models_{\Sigma} \varphi$, if $\mathcal{M}_i^{\text{C}} \models^{\text{C}}_{\Sigma_i^{\text{C}}} \varphi_i^{\text{C}}$ $(i = 1, 2)$ and $\mathcal{M}\ \models_{\Sigma} \varphi^{\text{rules}}$. In other words, a model satisfies a formula if the SW-models conform to the corresponding metamodels, and they fulfill the top transformation rules. The satisfaction relation $\mathcal{M} \models_{\Sigma} \varphi^{\text{rules}}$ is defined to hold if for all $\text{Rule}_i \in \text{top}.\ \mathcal{M}^{\text{FOL}}, \emptyset \models \text{Rule}_i$. We take $\emptyset$ as the empty variable assignment, since for rules it will be used only in the case of non top and explicit called rules.

Finally, given signatures Σ_i, a signature morphism $\sigma : \Sigma_1 \rightarrow \Sigma_2$, a Σ_2-model $\mathcal{M}$, a set of variables X_2, and a Σ_1-formula ψ with variables in $X_2|_\sigma$, the following satisfaction condition holds (see [26]): $\mathcal{M}|_\sigma \models_{\Sigma_1} \psi$ iff $\mathcal{M} \models_{\Sigma_2} \sigma(\psi)$. Thus, we can state that $\mathcal{I}^{\text{QVT}}$ consisting of the definitions given before, defines an institution. Complete definitions and proofs can be found in [26].

5.1 Running Example

The signature $\Sigma = \langle \Sigma_1^{\mathrm{C}}, \Sigma_2^{\mathrm{C}}, \Sigma^{\mathrm{FOL}} \rangle$ contains the signature Σ_1^{C} of the source metamodel, which is the one presented in Section 4.1, the signature Σ_2^{C} of the target metamodel, which is not shown here but can be derived in the same way, and a $FOL^{=}$ signature Σ^{FOL} with at least one sort for each type name in $\bigcup_i T(C_i)$ and a predicate for each property in $\bigcup_i P_i$.

A transformation between two SW-models is represented as a formula φ of the form $\langle \varphi_1^{\mathrm{C}}, \varphi_2^{\mathrm{C}}, \varphi^{\mathrm{rules}} \rangle$ such that, for example, φ_1^{C} is the formula introduced in Section 4.1, which represents the multiplicity constraints of the metamodel in Figure 1, φ_2^{C} is another formula representing the target SW-model (not shown here), and $\varphi^{\mathrm{rules}} = \langle \mathrm{Rules}, \mathrm{top} \rangle$ is the formula representing the transformation specification which has three relations named `PackageToSchema` $\in$ top, `ClassToTable` $\in$ top and `AttributeToColumn`.

As an example, the relation `PackageToSchema` is defined as follows:
`PackageToSchema` $= \langle \mathrm{VarSet}, \mathrm{Pattern}_i \ (i = 1, 2), \mathrm{when}, \mathrm{where} \rangle$ such that

- $\mathrm{VarSet} = \{\mathrm{pn}, \mathrm{p}, \mathrm{s}\}$ with $\mathrm{pn} \in X^{String}$, $\mathrm{p} \in X^{Package}$, and $\mathrm{s} \in X^{Schema}$.
- $\mathrm{Pattern}_1 = \langle E_1, A_1, Pr_1 \rangle$ with $E_1 = \{\mathrm{p}\}$, $A_1 = \emptyset$, and $Pr_1 =$ `name(p,pn)` (name is also a property in the source metamodel)
- $\mathrm{Pattern}_2 = \langle E_2, A_2, Pr_2 \rangle$ with $E_2 = \{\mathrm{s}\}$, $A_2 = \emptyset$, and $Pr_2 =$ `name(s,pn)`.
- $\mathrm{when} = \langle \emptyset, \emptyset \rangle$ and $\mathrm{where} = \langle \emptyset, \emptyset \rangle$.

A model $\mathcal{M} = \langle \mathcal{M}_1^{\mathrm{C}}, \mathcal{M}_2^{\mathrm{C}}, \mathcal{M}^{\mathrm{FOL}} \rangle$ can be composed by $\mathcal{M}_1^{\mathrm{C}} = (\mathcal{I}, \beta)$ as defined in Section 4.1, $\mathcal{M}_2^{\mathrm{C}} = (\mathcal{I}', \beta')$ is a target model not shown in this paper, and $\mathcal{M}^{\mathrm{FOL}}$ is a first-order structure. Binding of variables depends on the type of elements. If the variable is of a class, we have that the set of possible values coincides with the set of elements within the MOF institutions, e.g. $|\mathrm{p}| = V_{\mathrm{Package}} = \{p1\}$. However, if the variable is of a primitive type, we have than $V_t \subseteq |D|_t$ since transformation rules can use other elements besides those in the MOF institutions, for example those strings created using the type constructor ++, e.g. $|\mathrm{pn}| = \{Pac, Str, ID, ..., pk, tid, ..., ID++tid, ID++numb, ...\}$

We have that $\mathcal{M} \models_{\Sigma} \varphi$, if $\mathcal{M}_i^{\mathrm{C}} \models^{\mathrm{C}}_{\Sigma_i^{\mathrm{C}}} \varphi_i^{\mathrm{C}}$ $(i = 1, 2)$ and $\mathcal{M} \models_{\Sigma} \varphi^{\mathrm{rules}}$. We already showed that $\mathcal{M}_1^{\mathrm{C}} \models^{\mathrm{C}}_{\Sigma_1^{\mathrm{C}}} \varphi_1^{\mathrm{C}}$, and we prove in the same way that $\mathcal{M}_2^{\mathrm{C}} \models^{\mathrm{C}}_{\Sigma_2^{\mathrm{C}}} \varphi_2^{\mathrm{C}}$ for a valid SW-model. Thus, we need to prove that $\mathcal{M} \models_{\Sigma} \varphi^{\mathrm{rules}}$, and this holds if $\mathcal{M}^{\mathrm{FOL}}, \emptyset \models$ `ClassToTable`, and $\mathcal{M}^{\mathrm{FOL}}, \emptyset \models$ `PackageToSchema`.

As an example, we prove that $\mathcal{M}^{\mathrm{FOL}}, \emptyset \models$ `PackageToSchema` considering a valid target SW-model with only one schema (semantically represented as $s1$) having the same name as the package (semantically represented as Pac). We know that $|\mathrm{pn}| = V_{\mathrm{String}} = \{Pac, Str, Per, nul, ID, val, ...\}$, and $|\mathrm{p}| = V_{\mathrm{Package}} = \{p1\}$, so $|\{\mathrm{pn}, \mathrm{p}\}|$ is $\{(Pac, p1), (Str, p1), (ID, p1), (nul, p1), (Per, p1), (val, p1), ...\}$.

We also have that $|\mathrm{s}| = V_{\mathrm{Schema}} = \{s1\}$. Thus, the rule holds if

$$\forall\, \mu^1[\mathrm{pn}, \mathrm{p}] \in \{(Pac, p1), (Str, p1), (ID, p1), (Per, p1), (val, p1), (nul, p1), ...\},$$
$$(\ \mathcal{M}^{\mathrm{FOL}}, \mu^1 \models \mathrm{Pattern}_1 \rightarrow \exists\, \mu^2[\mathrm{s}] \in \{s1\},$$
$$(\ \mathcal{M}^{\mathrm{FOL}}, (\mu^1 \cup \mu^2) \models \mathrm{Pattern}_2\ \wedge \mathcal{M}^{\mathrm{FOL}}, (\mu^1 \cup \mu^2) \models \mathrm{where}))$$

For every μ^1[pn, p] different from $(Pac, p1)$ we have that Pattern_1 does not hold, since it depends on the predicate `name(p,pn)`. Thus, in these cases the implication holds. Now, in the case of $(Pac, p1)$, we have that Pattern_1 holds, and that the only possible value for s is $s1$. In this case, we also have that $\mathcal{M}^{\text{FOL}}, (\mu^1 \cup \mu^2) \models \text{Pattern}_2$ since the predicate `name(s,pn)` holds. Moreover, since the `where` clause is empty, $\mathcal{M}^{\text{FOL}}, (\mu^1 \cup \mu^2) \models$ where trivially holds. Finally, we conclude that $\mathcal{M}^{\text{FOL}} \models$ `PackageToSchema` indeed.

6 Conclusions and Future Work

In this paper we have defined institutions to represent the conformance relation between MOF models and metamodels, and the satisfaction of QVT-Relations check-only unidirectional transformations between pairs of models. These definitions neither depend on a shallow embedding of the languages by providing a syntactic translation into other logics, nor on the definition of specific institutions for each metamodel or model transformation. On the contrary, we defined a generic and minimal infrastructure within a theory which allows the definition of semantic-preserving translations from the MDE elements to potentially any logic defined as an institution, with the advantage that there is no need to maintain multiple formal representations of the same MDE elements.

Unlike MOF, we do not consider some constructions (e.g. aggregation, operations on classes) since they are elements not commonly used within transformations. We neither consider black-box operations or rule and transformation overriding within transformations since they are advanced features not commonly used in practice, nor keys definition since they are used for object creation not within the checking semantics. However, an inclusion of these elements within our institutions will strengthen the formal environment for MDE.

Our institutions contribute to the definition of a comprehensive formal environment for the verification of model transformations. We plan to define comorphisms from our institutions to a host logic and supplement this information with properties specified in the host logic. In particular, we have an initial formal definition of comorphism to CASL, the main language within Hets, and we are developing a first functional prototype of the running example using such definition as a way to test the main concepts. The native inclusion of MOF and QVT within Hets (by implementing the necessary Haskell code), as well as of the comorphisms, is within our medium-term goals.

Although we are for now focusing on MOF and QVT-Relations, we envision to extend the environment to support other transformation approaches.

References

1. Kent, S.: Model driven engineering. In: Butler, M., Petre, L., Sere, K. (eds.) IFM 2002. LNCS, vol. 2335, pp. 286–298. Springer, Heidelberg (2002)
2. Object Management Group: Meta Object Facility (MOF) 2.0 Core Specification. Specification Version 2.0 (2003)

3. Object Management Group: Meta Object Facility (MOF) 2.0 Query/View/Transformation. Final Adopted Specification Version 1.1 (2009)
4. Kurtev, I., Bezivin, J., Aksit, M.: Technological spaces: An initial appraisal. In: Intl. Symposium on Distributed Objects and Applications (2002)
5. Cengarle, M.V., Knapp, A., Tarlecki, A., Wirsing, M.: A heterogeneous approach to UML semantics. In: Degano, P., De Nicola, R., Meseguer, J. (eds.) Concurrency, Graphs and Models. LNCS, vol. 5065, pp. 383–402. Springer, Heidelberg (2008)
6. Mossakowski, T.: Heterogeneous specification and the heterogeneous tool set. Tech. Rep., Universitaet Bremen, Habilitation thesis (2005)
7. Goguen, J.A., Burstall, R.M.: Institutions: Abstract Model Theory for Specification and Programming. J. ACM 39(1), 95–146 (1992)
8. Object Management Group: Object Constraint Language. Formal Specification Version 2.2 (2010)
9. Calegari, D., Szasz, N.: Bridging techological spaces for the verification of model transformations. In: Conf. Iberoamericana de Software Engineering, Uruguay (2013)
10. Goguen, J.A., Rosu, G.: Institution morphisms. Formal Aspects of Computing 13, 274–307 (2002)
11. Mossakowski, T., Haxthausen, A.E., Sannella, D., Tarlecki, A.: Casl - the common algebraic specification language: Semantics and proof theory. Computers and Artificial Intelligence 22, 285–321 (2003)
12. Cengarle, M.V., Knapp, A.: An institution for UML 2.0 static structures. Tech. Rep. TUM-I0807, Institut für Informatik, Technische Universität München (2008)
13. James, P., Knapp, A., Mossakowski, T., Roggenbach, M.: Designing domain specific languages – A craftsman's approach for the railway domain using CASL. In: Martí-Oliet, N., Palomino, M. (eds.) WADT 2012. LNCS, vol. 7841, pp. 178–194. Springer, Heidelberg (2013)
14. Sannella, D., Tarlecki, A.: Foundations of Algebraic Specification and Formal Software Development. Springer (2012)
15. Beckert, B., Keller, U., Schmitt, P.: Translating the Object Constraint Language into first-order predicate logic. In: VERIFY Workshop, Denmark (2002)
16. Shan, L., Zhu, H.: Semantics of metamodels in UML. In: 3rd IEEE Symposium on Theoretical Aspects of Software Engineering, pp. 55–62. IEEE Computer Society (2009)
17. Rivera, J., Durán, F., Vallecillo, A.: Formal specification and analysis of domain specific models using Maude. Simulation 85(11-12), 778–792 (2009)
18. Boronat, A., Knapp, A., Meseguer, J., Wirsing, M.: What is a multi-modeling language? In: Corradini, A., Montanari, U. (eds.) WADT 2008. LNCS, vol. 5486, pp. 71–87. Springer, Heidelberg (2009)
19. Orejas, F., Wirsing, M.: On the specification and verification of model transformations. In: Palsberg, J. (ed.) Semantics and Algebraic Specification. LNCS, vol. 5700, pp. 140–161. Springer, Heidelberg (2009)
20. Bidoit, M., Hennicker, R., Tort, F., Wirsing, M.: Correct realizations of interface constraints with OCL. In: France, R.B. (ed.) UML 1999. LNCS, vol. 1723, pp. 399–415. Springer, Heidelberg (1999)
21. Boronat, A., Heckel, R., Meseguer, J.: Rewriting Logic Semantics and Verification of Model Transformations. In: Chechik, M., Wirsing, M. (eds.) FASE 2009. LNCS, vol. 5503, pp. 18–33. Springer, Heidelberg (2009)

22. de Lara, J., Guerra, E.: Formal Support for QVT-Relations with Coloured Petri Nets. In: Schürr, A., Selic, B. (eds.) MODELS 2009. LNCS, vol. 5795, pp. 256–270. Springer, Heidelberg (2009)
23. Anastasakis, K., Bordbar, B., Küster, J.M.: Analysis of model transformations via Alloy. In: 4th MoDeVVa Workshop, pp. 47–56 (2007)
24. Stenzel, K., Moebius, N., Reif, W.: Formal verification of QVT transformations for code generation. In: Whittle, J., Clark, T., Kühne, T. (eds.) MODELS 2011. LNCS, vol. 6981, pp. 533–547. Springer, Heidelberg (2011)
25. Guerra, E., de Lara, J.: An algebraic semantics for QVT-relations check-only transformations. Fundamenta Informaticae 114, 73–101 (2012)
26. Calegari, D., Szasz, N.: Institution-based semantics for MOF and QVT-Relations (extended version). Tech. Rep. 13-06, InCo-PEDECIBA (2013) ISSN 0797-6410, `http://www.fing.edu.uy/inco/pedeciba/bibliote/reptec/TR1306.pdf`