Institution-based Foundations for Verification in the Context of Model-Driven Engineering

Daniel Calegari\textsuperscript{a,}\textsuperscript{*}, Nora Szasz\textsuperscript{b}

\textsuperscript{a}Facultad de Ingeniería, Universidad de la República, Montevideo 11300, Uruguay
\textsuperscript{b}Facultad de Ingeniería, Universidad ORT Uruguay, Montevideo 11100, Uruguay

Abstract

A separation of duties between software developers is usually proposed to cope with formal verification issues within the Model-Driven Engineering (MDE) paradigm. MDE experts are responsible for the definition of models and model transformations, while formal verification experts conduct the verification process. This schema should be aided by (semi)automatic translations from the MDE elements to their formal representation in the potentially many semantic domains used for verification, and also by translations between these domains. Translations may be useful to perform a heterogeneous verification, i.e. using different domains for the verification of each part of the whole problem, and also to integrate MDE elements with the specification and verification of other traditional software artifacts. However, this schema requires formal foundations allowing the representation of the MDE elements in such a way that it is possible to ensure that translations are semantic-preserving. The aim of this paper is to present a formalization of the MDE elements using the Theory of Institutions. We provide institutions for the representation of MDE elements based on the MOF and QVT-Relations standards. We also show how the theory assists with these requirements for the definition of an environment for the formal verification using heterogeneous verification approaches.

\textsuperscript{c}© 2015 Published by Elsevier Ltd.

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

1. Introduction

The Model-Driven Engineering paradigm (MDE, [1]) refers to the systematic use of models as primary engineering artifacts throughout the engineering process. In particular, the paradigm 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. It also encompasses other engineering efforts of a complete software engineering process, such as maintenance and reverse engineering. This approach tends to improve efficiency on the construction process and the trustworthiness of the resulting products.

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. Models are defined from metamodels, i.e. models which introduce the syntax and semantics of certain domain-specific kind of models. When a model is structurally compliant with respect

\textsuperscript{*}Corresponding author. Tel.: +598-27114244; fax: +598-27110469.

Email addresses: dcalegar@fing.edu.uy (Daniel Calegari), szasz@ort.edu.uy (Nora Szasz)
to its metamodel, their relation is called \textit{conformance} [2]. In some cases, there are conditions (called invariants) that cannot be captured by the structural rules of this language, in which case the language is supplemented with another one, e.g. the Object Constraint Language (OCL, [3]). A model transformation (or just transformation from now on) is basically the automatic generation of a target model from a source model, according to a transformation definition, i.e. a set of rules that together describe how a model in the source language can be transformed into a model in the target language [4]. The Object Management Group (OMG) conducted a standardization process of languages for MDE. They defined the MetaObject Facility (MOF, [5]) as the language for metamodeling, and three transformation languages with different transformation approaches. The Query/View/Transformation Relations (QVT-Relations, [6]) is one of those languages and follows a relational approach which consists on defining transformations as declarative relations between source and target elements.

The quality of the whole MDE process strongly depends on the quality of the models and model transformations. It is increased by verification of the generated models, and of the model transformations, at early development stages. In some cases formal methods (i.e. mathematically based techniques) arise as a tool for strengthening verification results. The specification and verification of an MDE-built system has some parallelism with traditional software systems. The formal treatment of a problem requires some notation with formal semantics, along with a deductive system for reasoning. This is often considered difficult to apply and requires significant mathematical experience, which leads to a mismatch problem between software engineering expectations and formal methods possibilities.

To cope with this situation, a separation of duties between software developers is usually proposed, giving rise to different technological spaces [7], 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 experts conduct the verification process, often aided by some (semi)automatic generation process which translates the MDE elements to their formal representation into the domain used for verification purposes. The formal representation is usually defined in a unified semantic domain (e.g. [8, 9, 10]), having tools for conducting the verification process and for retrieving some feedback to the MDE experts. However, the use of a mathematical formalism serving as a unique semantic basis for verification can be quite restrictive considering that there are several alternatives which depend on those properties that must be addressed in each specific case [11]. Depending on the problem, it can be useful to have a representation of MDE elements in different logical domains. To achieve this, one possibility is to generate partial or complete formal representations of elements from the MDE technological space in different formal domains, and then “connect” these specifications via translations between the logical domains. These connections may be useful to perform a heterogeneous verification, i.e. using different domains for the verification of each part of the whole problem [12], and also to integrate MDE elements with the specification and verification of other traditional software artifacts [13]. The currently used semantic domains are not well adapted for the definition of such schema, since it requires the formal representation of the MDE elements in such a way that it is possible to ensure that translations between these domains are semantic-preserving. If the semantics is not preserved then it is no possible to ensure that proofs in different domains (e.g. first-order or rewriting logics) are sound with respect to the original problem.

The aim of this paper is to present a formalization of the MDE elements (based on the MOF and QVT-Relations standards) using the Theory of Institutions [14]. We focused on more specifically structural semantic aspects instead of dynamic semantics aspects of MDE elements. The theory provides ways of representing the syntax and semantics of the MDE elements: models, metamodels, the conformance relation between them, and model transformations as institutions, in some consistent and interdependent way. An institution abstracts the representation of any specification language, providing mechanisms for the definition of semantic-preserving translations between semantic domains (which are also represented as institutions).

This paper is a substantially extended and thoroughly revised version of [15]. Additional material includes:

- a detailed and improved formalization of the institutions, splitting them into standard versions for the representation of the problem and proof-theoretical extensions devised to be used within a proof environment
- an extended formal treatment of the conformance relation, with a discussion about the inclusion of an institution for OCL and for model typing
- a discussion about the definition of an institution-based environment for the formal verification of MDE elements using heterogeneous approaches [16], together with a detailed discussion with respect to related work
The remainder of the paper is structured as follows. In Section 2 we present the basic notions of the Theory of Institutions which will be useful for the theoretical understanding of the following sections. For a more detailed introduction to the topic refer to [17, 18]. In the following sections, we provide an institution-based formalization of the MDE elements which is the main concern of this paper: in Section 3 we provide formal definitions of institutions for representing models, metamodels and the MOF-based conformance relation between them, in Section 4 we define an institution for QVT-Relations model transformations, and in Section 5 we define an extension of these institutions in order to be used within a proof environment. In Section 6 we introduce how these formal foundations can be used for the definition of an environment for the formal verification of MDE elements using heterogeneous verification approaches. In Section 7 we summarize related works. Finally, in Section 8 we present some conclusions and guidelines for future work.

2. A Quick Look at Institutions

The concept of Institution formalizes the notion of “logical system”, which can be seen as a set of principles for some form of sound reasoning [17]. Many different logics, as first-order, modal and rewriting, have been shown to be institutions [18]. An interesting aspect is that within most specification formalisms there is a logical system allowing the user to write axioms describing the properties of the software system to be developed. In this sense, the notion of institution can be used to represent any specification language since it provides ways for representing the syntax and semantics of the language, as well as the relation between them by means of a satisfaction relation. Examples of this are the institutions defined for UML languages [13].

An institution consists of vocabularies (called signatures) for constructing sentences in a logical system. A model (also called interpretation) provides semantics by assigning interpretations to the elements in the signature. We can allow a change of interpretation defining the notion of homomorphism, which consists of a mapping of elements between two models. Institutions also define formal translations (called signature morphisms) between signatures, allowing many different vocabularies at once. Since signatures can change through signature morphisms, we need to translate sentences and models accordingly. Sentences are translated along signature morphisms since symbols must be replaced in each sentence conforming to the signature morphism. In the case of models, they are translated in the opposite direction of signature morphisms, i.e. a model providing semantics to the target signature of a signature morphism is reduced to a model providing semantics to the source signature (target signatures potentially have more elements than source signatures). Finally, there is a satisfaction relation of sentences by models, such that when a signature is changed (by a signature morphism), satisfaction of sentences by models changes consistently. The formal definition of an institution relies on Category Theory [19].

Definition 1 (Institution). An institution \( \mathcal{I} \) (as defined in [18]) consists of:

1. a category \( \text{Sign}_\mathcal{I} \)\(^1\) of signatures:
2. a functor \( \text{Sen}_\mathcal{I} : \text{Sign}_\mathcal{I} \to \text{Set} \), giving a set \( \text{Sen}(\Sigma) \) of \( \Sigma \)-sentences for each signature \( \Sigma \in \text{Sign}_\mathcal{I} \)\(^2\) and a function \( \text{Sen}_\mathcal{I}(\sigma) : \text{Sen}_\mathcal{I}(\Sigma_1) \to \text{Sen}_\mathcal{I}(\Sigma_2) \) translating \( \Sigma_1 \)-sentences to \( \Sigma_2 \)-sentences for each signature morphism \( \sigma : \Sigma_1 \to \Sigma_2 \);
3. a functor \( \text{Mod}_\mathcal{I} : \text{Sign}_\mathcal{I}^{op} \to \text{Cat} \)\(^3\), giving a category \( \text{Mod}(\Sigma) \) of \( \Sigma \)-models for each signature \( \Sigma \in \text{Sign}_\mathcal{I} \) and a function \( \text{Mod}_\mathcal{I}(\sigma) : \text{Mod}_\mathcal{I}(\Sigma_2) \to \text{Mod}_\mathcal{I}(\Sigma_1) \) translating \( \Sigma_2 \)-models to \( \Sigma_1 \)-models (and \( \Sigma_2 \)-morphisms to \( \Sigma_1 \)-morphisms) for each signature morphism \( \sigma : \Sigma_1 \to \Sigma_2 \);
4. for each signature \( \Sigma \in \text{Sign}_\mathcal{I} \), a satisfaction relation \( \models_{\mathcal{I},\Sigma} \subseteq \text{Mod}_\mathcal{I}(\Sigma) \times \text{Sen}_\mathcal{I}(\Sigma) \) such that for any signature morphism \( \sigma : \Sigma_1 \to \Sigma_2 \) the translation \( \text{Mod}_\mathcal{I}(\sigma) \) of models and \( \text{Sen}_\mathcal{I}(\sigma) \) of sentences preserve the satisfaction relation, that is, for any \( \varphi \in \text{Sen}_\mathcal{I}(\Sigma_1) \) and \( M_2 \in \text{Mod}_\mathcal{I}(\Sigma_2) \):

\[
M_2 \models_{\mathcal{I},\Sigma_2} \text{Sen}_\mathcal{I}(\sigma)(\varphi) \iff \text{Mod}_\mathcal{I}(\sigma)(M_2) \models_{\mathcal{I},\Sigma_1} \varphi
\]

\(^1\)We often omit the subscript \( \mathcal{I} \)
\(^2\)|\( |C| \) is the collection of objects of a category \( C \)
\(^3\)\( \text{Sign}_\mathcal{I}^{op} \) is the opposite category of the category \( \text{Sign}_\mathcal{I} \)
The definition of an institution can be extended to consider not only an individual signature, but a theory, i.e. a pair $T = (\Sigma, \Psi)$ consisting of a signature $\Sigma$ and an arbitrary set of axioms, which are $\Sigma$-sentences. From a model-theoretic point of view, it is possible to define the notion of logical consequence or semantic entailment as follows.

**Definition 2 (Semantic entailment).** Given an institution $I = (\Sigma, \Psi, \cong)$, a set of $\Sigma$-sentences $\Psi$ and a $\Sigma$-sentence $\varphi$, we say $\Psi \models_\Sigma \varphi$ iff for all $\Sigma$-models $M$, we have

$$M \models_\Sigma \Psi \text{ implies } M \models_\Sigma \varphi$$

Here, $M \models_\Sigma \Psi$ means that $M \models_\Sigma \psi$ for each $\psi \in \Psi$. Moreover, it is possible to extend an institution from a proof-theoretic point of view by defining a logic in some way compatible with semantic entailment.

**Definition 3 (Logic).** A logic $LOG = (\Sigma, \Psi, \cong, \vdash)$ is an institution $(\Sigma, \Psi, \cong, \models_{\Sigma})$ equipped with an entailment system $\vdash$ that is, a relation between sentences $\vdash_{\Sigma} \subseteq P(Sen(\Sigma)) \times Sen(\Sigma)$ for each $\Sigma \in [\Sigma_0]$, such that some properties are satisfied. In particular, the relation must be sound (all provable statements are true), i.e. $\Psi \models_{\Sigma} \varphi$ implies $\Psi \models_{\Sigma} \varphi$. In some cases the entailment system can be complete: $\Psi \models_{\Sigma} \varphi$ implies $\Psi \models_{\Sigma} \varphi$, i.e. all true statements are provable. Entailment is typically defined via a system of finitary derivation rules, giving the notion of proof that is absent when the institution is considered on its own, even if $\vdash$ coincides with semantic entailment.

3. Institutions for Conformance

The conformance relation can be defined in terms of structural and semantical (or non-structural) conformance [20]. A model (from now on SW-model, to avoid conflicts with a model of an institution) is structurally conformant with respect to a metamodel if it is well-typed with respect to the metamodel and it also satisfies the multiplicity constraints. Semantical conformance requires, in addition to structural conformance, that the SW-model satisfies the invariants of its metamodel, which are specified using a supplementary constraints language, e.g. OCL.

The MOF standard defines Essential MOF (EMOF) which is a subset of MOF allowing simple metamodels to be defined. In few words, an EMOF metamodel defines classes which can belong to a hierarchical structure. Some of them may 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 constrains the number of elements that can be related through the property, and it can be related with another property (known as its opposite) if the property corresponds to a bidirectional association between two classes.

In order to present our formalization we shall consider the following example. The metamodel in Figure 1a defines UML class diagrams, where classifiers (classes and primitive types as string, boolean, integer, etc.) are contained in packages (association contains). Classes can contain attributes (association has) and may be declared as persistent (kind = ‘Persistent’), whilst attributes have a type that is a primitive type (association typeOf). Notice that a class must contain only one or two attributes (multiplicity 1..2), and also that the Classifier class is not abstract. We decided to handle these aspects differently from UML class diagrams in order to have a more complete example. The metamodel has some invariants, as for example that there cannot be two Classifier with the same name within the same Package. This invariant can be expressed using OCL as follows.

```ocl
classifier->forall( c1, c2 : Classifier | c1 <> c2 implies c1.name <> c2.name)
```

The SW-model in Figure 1c is structurally conformant to the metamodel depicted in Figure 1a. It is 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 which is a primitive type. Moreover, it is semantically conformant to the metamodel since it obeys the invariant. Moreover, the relational SW-model in Figure 1d is structurally conformant to the Relational diagrams metamodel depicted in Figure 1b. Every schema contains a number of tables and each table has a number of columns. Each column has a name and a kind, which can be the primary keys of the corresponding table. Relational diagrams will be used in Section 4 as part of the running example.

In what follows we present institutions for the representation of SW-models, metamodels and the conformance relation between them. We first define an institution for the structural conformance relation which is based on the
institutions for UML class diagrams presented in [21, 22], but adapted for representing metamodels. This institution considers the satisfaction of typing requirement by construction, as well as multiplicity constraints checking as part of the satisfaction relation. Nevertheless, we then discuss how to shape an institution for model typing. Finally, we shape an institution for semantical conformance with respect to OCL invariants. Any of these proposals use already existent formalizations at the level of institutions, i.e. formal representations are taken from existent works and fitted to define every element of the institutions with their corresponding proofs.

3.1. Structural Conformance

We define an institution $I^M$ for the structural conformance relation between SW-models and metamodels specified with a simplified version of MOF (which we call CSMOF). As said, it is based on the institutions for UML class diagrams presented in [21, 22], but adapted for representing metamodels. Unlike [21], in our definition there are no derived relations, the signature has an explicit representation of abstract classes, and there are only binary properties. Derived relations and n-ary properties are not used within transformations, and abstract classes where not considered in the former works. Moreover, unlike MOF, we do not consider aggregation, uniqueness and ordering properties within a property end, operations on classes, or packages. Aggregation and operations are not used within transformations, whilst packages are just used for organizing metamodel elements (since they can be considered syntactic sugar). Although uniqueness and ordering properties are neither commonly used, their future inclusion will improve the institution.

For the definition of the institution we follow the schema in Figure 2. From any metamodel we can derive a signature with a representation of types and properties (attributes and associations). Formulas represent multiplicity constraints determining whether the number of elements in a property end is bounded (upper and/or lower). A model contains a semantic representation of a SW-model. Given a model representing a SW-model, the satisfaction relation applied to set of multiplicity constraints derived from the metamodel answers the following question: does the SW-model structurally conforms to the metamodel?
Figure 2: The conformance relation as an institution

Besides this institution is an updated version of previous work, we just present here its main components: signatures, formulas, models and the satisfaction relation. Complete definitions and proofs can be found in [16].

A signature represents a metamodel, i.e. it defines hierarchical related classes, primitive types and type constructors. For such reason we first introduce class hierarchies.

Definition 4 (Class hierarchy and type extension). A class hierarchy is represented as a partial order \( C = (C, \leq_C) \) where \( C \) is a set of class names, and \( \leq_C \subseteq C \times C \) is the subclass (inheritance) relation. By \( T(C) \) we denote the type extension of \( C \) by primitive types (e.g. Boolean and String) and type constructors (e.g. List and Set). \( T(C) \) is likewise a class hierarchy \( (T(C), \leq_{T(C)}) \) with \( C \subseteq T(C) \) and \( \leq_C \subseteq \leq_{T(C)} \), which is closed with respect to types, and “downwards” closed with respect to type constructors.

We can consider a fixed set of primitive types and type constructors similar to those defined for the OCL [3]. As in [22], in order to provide generic access to primitive types and type constructors, we treat these as built-in with a standard meaning. All other classes are assumed to be inhabited, i.e., to contain at least one object. However, unlike [22] in which the existence of an object \( \textit{null} \) is assumed, we impose that if \( c \) is abstract then there exists another \( c' \) in the hierarchy such that \( c' \leq_{T(C)} \cdots \leq_{T(C)} c \) and \( c' \) has at least one object.

Definition 5 (CSMOF signature). A CSMOF signature \( \Sigma = (C, \alpha, P) \) declares:

- a finite class hierarchy \( C = (C, \leq_C) \) extended with a subset \( \alpha \subseteq C \) denoting abstract classes
- a properties declaration (attributes and associations) \( P = (R, P) \) where \( R \) is a finite set of role names with a default role name “\( . \)”, and \( P \) is a finite set of properties of the form \( \langle r_1 : c_1, r_2 : c_2 \rangle \) with \( r_1, r_2 \in R, c_1, c_2 \in T(C) \), such that for any class or type name \( c \in T(C) \), the role names of the properties in which any \( c' \leq_{T(C)} c \) is involved are all different, i.e. if \( \langle r_1 : c_1, r_2 : c_2 \rangle \) and \( \langle s_1 : d_1, s_2 : d_2 \rangle \) are properties in \( P \) and \( c_k = d_l \in T(C) \), then \( r_i \neq s_j \) for any \( i \neq k \) and for any \( j \neq l \) (\( 1 \leq i \leq 2, 1 \leq j \leq 2 \))

Any property declaration \( \langle r_1 : c_1, r_2 : c_2 \rangle \in P \) represents a MOF property and its opposite, such that the type of the property, as well the type in the opposite side represents its owned class. The default role name “\( . \)“ is used if a property has no opposite. For example, in the case of an attribute \( r \) of type \( d \) in the type \( c \), the property declaration will be \( \langle . : c, r : d \rangle \), and in the case of an unidirectional association, the role in the opposite side of the arrow must also be the default role name “\( . \)“.

From the class metamodel in Figure 1a we derive the signature \( \Sigma = (C, \alpha, P) \) with \( C = (C, \leq_C) \) and \( P = (R, P) \), such that:
\[ C = \{ \text{UMLModelElement, Package, Classifier, PrimitiveDataType, Attribute, Class} \} \]

and \( T(C) \) also contains type \( \text{String} \)

\[ \leq_C = \{ \text{Package} \leq_C \text{UMLModelElement}, \text{Attribute} \leq_C \text{UMLModelElement}, \text{Classifier} \leq_C \text{UMLModelElement}, \text{Class} \leq_C \text{Classifier, PrimitiveDataType} \leq_C \text{Classifier} \} \]

\[ \alpha = \{ \text{UMLModelElement} \} \]

\[ R = \{ \text{namespace, elements, type, owner, attribute, name, kind} \} \]

\[ P = \{ \text{namespace} : \text{Package, elements} : \text{Classifier}, \text{namespace} : \text{String}, \text{type} : \text{Classifier, owner} : \text{Class}, \text{name} : \text{String, kind} : \text{PrimitiveDataType} \} \]

Formulas represent multiplicity constraints, i.e. determining whereas the number of elements in a property end is bounded (upper and/or lower).

**Definition 6 (CSMOF formula).** Given a signature \( \Sigma = (C, \alpha, P) \) with \( C = (C, \leq_C) \) and \( P = (R, P) \), a \( \Sigma \)-formula representing a multiplicity constraint is defined by the following grammar:

\[
\Phi ::= \#\Pi = N \mid N \leq \#\Pi \mid \#\Pi \leq N
\]

\[
\Pi ::= C \bullet R
\]

The \#-expressions return the number of links in a property when some role is fixed. We use \( \bullet \) as an operator representing the selection of the elements linked with an element of class \( c \in C \) through role \( r \in R \); there must exist a property \( (r' : c, r : d) \) or \( (r : d, r' : c) \) in \( P \).

The set of formulas \( \varphi \) corresponding to the metamodel in Figure 1a is defined by:

\[ \varphi = \{ \#(\text{UMLModelElement} \bullet \text{name}) = 1, \#(\text{UMLModelElement} \bullet \text{kind}) = 1, 1 \leq \#(\text{Class} \bullet \text{attribute}), \#(\text{Class} \bullet \text{attribute}) \leq 2, \#(\text{Attribute} \bullet \text{type}) = 1, \#(\text{Attribute} \bullet \text{owner}) = 1, \#(\text{Classifier} \bullet \text{namespace}) = 1 \} \]

The formulas allow representing any kind of multiplicity, e.g. that the number of attributes of a class has a lower bound of one element, and also and upper bound of two. In the other cases there are only lower bounds defined. If a property end has no associated formula, it means that there is no bound defined for it.

An interpretation (or model) contains a semantic representation for a SW-model, i.e. objects and links. For such reason we need to define the notion of object domain with respect to a class hierarchy.

**Definition 7 (Object domain and value extension).** Given a class hierarchy \( C = (C, \leq_C) \), a \( C \)-object domain \( O \) is a family \( \{O_c\}_{c \in C} \) of sets of object identifiers verifying \( O_{c1} \subseteq O_{c2} \) if \( c1 \leq_C c2 \). Given a type extension \( T \), the value extension of a \( C \)-object domain \( O = (O_c)_{c \in C} \) by primitive values and value constructions, which is denoted by \( V^T_C(O) \), is a \( T(C) \)-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, in particular, if \( c1 \leq_C c \) and \( c2 \leq_C c \), then \( O_{c1} \cap O_{c2} = \emptyset \).

**Definition 8 (CSMOF interpretation).** Given a signature \( \Sigma = (C, \alpha, P) \) with \( C = (C, \leq_C) \) and \( P = (R, P) \), a \( \Sigma \)-interpretation \( I \) consists of a tuple \( (V^I_C(O), A) \) where

- \( V^I_C(O) = (V_c)_{c \in T(C)} \) is a \( T(C) \)-object domain
- \( A \) contains a relation \( \langle r_1 : c_1, r_2 : c_2 \rangle \subseteq V_{c_1} \times V_{c_2} \) for each relation name \( \langle r_1 : c_1, r_2 : c_2 \rangle \in P \) with \( c_1, c_2 \in T(C) \)
- \( c2 \in \alpha \) implies \( O_{c2} = \bigcup_{c1 \leq_C c_2} O_{c1} \)

An interpretation \( I \) can have one element for each type in the signature as follows:

- \( A \) a \( T(C) \)-object domain consisting of

\[
\begin{align*}
V_{\text{Class}} &= \{ c \}, & V_{\text{PrimitiveDataType}} &= \{ \text{pdt1} \}, & V_{\text{Classifier}} &= \text{classification} \cup V_{\text{PrimitiveDataType}} \\
V_{\text{Package}} &= \{ \text{p1} \}, & V_{\text{Attribute}} &= \{ \text{a1} \}, & V_{\text{UMLModelElement}} &= \text{classifier} \cup V_{\text{Package}} \cup V_{\text{Attribute}} \\
V_{\text{String}} &= \{ \text{facs, str, per, nul, ID, val} \}
\end{align*}
\]
A set $A$ consisting of relations:

- $\langle \cdot : \text{UMLModelElement}, \text{name} : \text{String}\rangle^I = \{(p1,Pac),(c1,ID),(c2,nul),(a1,\text{val})\}$
- $\langle \cdot : \text{UMLModelElement}, \text{kind} : \text{String}\rangle^I = \{(p1,nul),(c1,\text{Per}),(a1,nul),(pdt1,nul)\}$
- $\langle \text{namespace} : \text{Package}, \text{elements} : \text{Classifier}\rangle^I = \{(p1,c1),(p1,pdt1)\}$
- $\langle \text{attribute} : \text{Attribute}, \text{owner} : \text{Class}\rangle^I = \{(a1,c1)\}$
- $\langle \cdot : \text{Attribute}, \text{type} : \text{PrimitiveDataType}\rangle^I = \{(a1,pdt1)\}$

Given a $\Sigma$-interpretation, the evaluation of an expression $c_i \bullet r_i$, associated to a property $\langle r_1 : c_1, r_2 : c_2 \rangle$, with respect to the interpretation gives a set of sets of pairs of semantic elements connected through property $\langle r_1 : c_1, r_2 : c_2 \rangle$, grouped by the semantic elements with type $c_i$. Note that this set can be empty if the element with type $c_i$ is not connected with any other. Formally:

**Definition 9 (Evaluation of properties).** Given a signature $\Sigma = (C, \alpha, P)$ with $C = (C, \leq C)$ and $P = (R, P)$, a $\Sigma$-interpretation $I = (V^I(O), A)$, and a property $\langle r_1 : c_1, r_2 : c_2 \rangle \in P$, we define the evaluation of $c_i \bullet r_j$ as follows:

$$(c_i \bullet r_j)^I = \{\{t \in \langle r_1 : c_1, r_2 : c_2 \rangle^I \mid \pi_r(t) = o \mid o \in V_{c_i}\} | i = 1, 2\}.$$

Consider the property $\langle \text{namespace} : \text{Package}, \text{elements} : \text{Classifier}\rangle$ representing that a package contains classifiers. This interpretation evaluates $(\text{Classifier} \bullet \text{namespace})^I$ as the set $\{\{(p1,c1)\},\{(p1,pdt1)\}\}$ since there is only one object with role namespace which is the package object $p1$, and there are two elements ($c1$ and $pdt1$) in the opposite side of the property to group by.

We must express that a SW-model conforms to a metamodel if it is well-typed and it also satisfies its multiplicity constraints. Well-typing holds by construction, since the interpretation representing a SW-model respects the signature which defines types within the metamodel. The satisfaction of multiplicity constraints (formulas) by a SW-model (interpretation) is thus the main concern of the satisfaction relation.

**Definition 10 (CSMOF satisfaction relation).** Given a signature $\Sigma = (C, \alpha, P)$ with $C = (C, \leq C)$ and $P = (R, P)$, a $\Sigma$-formula $\varphi$ representing a multiplicity constraint and a $\Sigma$-interpretation $I$, the interpretation satisfies $\varphi$, if one of the following holds:

- $\varphi$ is $\#(c \bullet r) = n$ and $|S| = n$ for all $S \in (c \bullet r)^I$
- $\varphi$ is $n \leq \#(c \bullet r)$ and $n \leq |S|$ for all $S \in (c \bullet r)^I$
- $\varphi$ is $\#(c \bullet r) \leq n$ and $|S| \leq n$ for all $S \in (c \bullet r)^I$

This means that for any object of class $c$, the number of elements within $I$ related through the role $r$ (of a property of the class) satisfies the multiplicity constraints.

This definition can be trivially extended for a set of formulas $\Phi$, as follows: $I \models_\Sigma \Phi$ if $I \models_\Sigma \varphi \forall \varphi \in \Phi$.

Back to the example, we can check that $I \models_\Sigma \varphi$ for every formula $\varphi$ representing a multiplicity constraints defined before.

- $\#(\text{UMLModelElement} \bullet \text{name}) = 1$ and $|S| = 1$ for all $S \in (\text{UMLModelElement} \bullet \text{name})^I = \{(p1,Pac),(c1,ID),(a1,\text{val})\}$
- $\#(\text{UMLModelElement} \bullet \text{kind}) = 1$ and $|S| = 1$ for all $S \in (\text{UMLModelElement} \bullet \text{kind})^I = \{(c1,\text{Per}),(pdt1,nul),(a1,nul)\}$
- $\#(\text{Classifier} \bullet \text{namespace}) = 1$ and $|S| = 1$ for all $S \in (\text{Classifier} \bullet \text{namespace})^I = \{(p1,c1),(p1,pdt1)\}$
- $\#(\text{Attribute} \bullet \text{owner}) = 1$ and $|S| = 1$ for all $S \in (\text{Attribute} \bullet \text{owner})^I = \{(a1,c1)\}$
- $\#(\text{Attribute} \bullet \text{type}) = 1$ and $|S| = 1$ for all $S \in (\text{Attribute} \bullet \text{type})^I = \{(a1,pdt1)\}$
the form of the representation of metamodels and SW-models. The idea is based on representing elements as graphs, i.e. structures $\text{Type \ Graphs \ with \ Node \ Type \ Inheritance}$ [24] which are commonly used in the field of graph transformation for the representation of attributed graphs. A type graph can be extended with an inheritance relation and a set of abstract node types [24]. The typing itself is depicted by a graph morphism between the initial graph and the type graph. These basic settings can be improved. In particular, it is possible to define attributed graphs which are graphs with attributes associated to nodes and arrows, and attributed type graphs which are also graphs defining types with a correspondence with many possible attributed graphs. A type graph can be extended with an inheritance relation and a set of abstract node types [24].

In conclusion, in analogy with SW-models which conform to a metamodel, we have attributed graphs typed with respect to an attributed type graph with inheritance. The typing relation within the structural conformance relation can be stated as a graph morphism between an attributed graph and an attributed type graph. In [25] there is an example of the formal definition of the abstract syntax of UML class and sequence diagrams based on typed attributed graph transformation with inheritance. We can assume that the typing problem is completely resolved by the existence of such graph morphism.

In order to define an institution for model typing, we need to put metamodels and SW-models at both sides of such relation, for example representing metamodels as interpretations (as in CSMOF) and SW-models as formulas. However, an important aspect to consider is that metamodels (interpretations) and SW-models (formulas) must not be constrained by a common set of types (within the signature as in CSMOF). In this sense, it can be no typing relation between them, which is exactly what the satisfaction relation must define. Leaving aside the details, we can represent attributed graphs (SW-models) as formulas, and attributed type graphs with inheritance (metamodels) as interpretations. We have a fixed signature which has the sorts (e.g. for vertices and edges) and functions (e.g. relating vertices and edges) for the definition of any attributed (type) graph. A signature morphism is just a renaming of the fixed sorts and functions. Homomorphisms and reducts are pointless since the signature morphism is just a renaming. Finally, the satisfaction relation is expressed as the existence of a graph morphism between the attributed graph and the type attributed graph with inheritance.

\[ 1 \leq \#(\text{Class} \bullet \text{attribute}) \leq |S| \text{ for all } S \in \{\text{Class} \bullet \text{attribute}\}^T = \{\{(a,1,1)\}\} \]

In [16] there are complete definitions of the missing elements of the institution: signature morphisms, homomorphisms and reduct. Moreover, there are complete proofs that signatures and signature morphisms define a category, that there is a functor from this category to the category of sets of formulas and their translations, that interpretations and homomorphisms define a category, that the reduct defines a functor and thus there is a functor giving a category of interpretations for each signature and a functor defined by the reduct, and that the satisfaction condition holds. With this last result we can state that $I^M$ consisting of signatures, morphisms, formulas, interpretations, reducts, homomorphisms, and the satisfaction relation, defines an institution.
3.3. Semantical Conformance

Semantical conformance is not addressed by the CSMOF institution. However, we would rather have an institution for OCL to use it not only for expressing general constraints on metamodels, but also for constraining QVT-Relations transformation rules which may also contain arbitrary boolean OCL expressions. In what follows we discuss the complexity of defining such institution and we shed some light on its definition.

The semantics of OCL defined in the standard leads to many interpretations and problems which were widely studied in the last decade. Most proposals define the semantics in terms of a shallow embedding of OCL into others formal languages, e.g. rewriting logic [26] and first-order logic [27, 28, 29], in a search for automatic checking tools. In most cases, formal semantics are given for an expressive subset of the language but not for the complete language. The most problematic aspect for defining OCL semantics is the fact that language is a three-valued logic, i.e. beyond the notion of truth or falsity of a constraint there exists the notion of undefinedness, thus many proposals do not deal with this notion. Moreover, in the last version of OCL [3] another element was added (null) which introduces a fourth possible value. In any case, these proposals do not ensure that the mappings are correct with respect to the OCL formal semantics, which in fact is not completely defined. If OCL were defined as an institution in which the semantics is given in terms of the satisfaction relation, the embedding could be represented as a semantic-preserving relation between institutions (as we discuss in Section 6).

We have two alternatives to handle OCL in our institutional settings. The first alternative is to use any of the existent embeddings, assuming their correctness with respect to OCL semantics. Since the referred proposals are defined in terms of logics which are already defined as institutions, we can use those proposals in connection with our institutions. For this to be possible, we need to study how our institutions can also be translated into those logics, and how they can be connected to the OCL embedding to work together. In Section 6 we explain what it means to “move” between institutions. The second alternative is to extend our CSMOF institution to handle OCL expressions. In few words, signatures and interpretations will be the same as those already defined for CSMOF since they define a concrete metamodel and SW-model, respectively. In addition to multiplicity constraint formulas, there will be OCL expressions referencing elements within the metamodel (signature). Notice that multiplicity constraints can also be expressed in OCL. Those expressions must be satisfied with respect to the SW-model represented in the institution model. The satisfaction relation needs to be handled in a different way for considering this fourth-valued logic. Fortunately, in [30] the authors define an institutional basis to combine many-valued logics with other logical systems. In particular they define a generic institution which can be used as a semantic oriented framework for defining in a uniform way new concrete many-valued logical systems. In resume, we can adapt (or define) some logic independent semantics of OCL to this institutional framework to shape our desired institution. We can then define translations of our elements as a whole into other logics.

Besides further studies should be conducted in both cases, from a formal perspective we can assume the existence of an expressions language which can be defined as an institution. In particular we consider a generic institution $I^\text{E}$ as an expressions language, which can be instantiated for example with an institution for first-order logic with equality ($FOL^=$) as defined in [18, 9], which made the expressions language equivalent to those defined in [27, 28, 29]. This generic institution will be used as a supporting institution in Section 4.

4. An Institution for QVT-Relations

A QVT-relational transformation definition can be basically described as a set of interconnected relations (we will refer to them as rules indistinctly). Relations 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. Relations define source/target domain patterns which are used to find matching elements in a SW-model. Relations can also contain when and where clauses. A when clause specifies the conditions under which the relationship holds, whilst the where clause specifies the condition that must be satisfied by all SW-model elements participating in the relation, and it may constrain any of the variables in the relation and its domains.

Consider the following example which is a simplified version of the well-known Class to Relational transformation [6]. The transformation basically describes how persistent classes within a package are transformed into tables within a schema. The relation PackageToSchema states that any UML package is mapped into a schema. The relation ClassToTable states that classes marked as persistent are mapped into tables with the same
name, a primary key and an identifying column, such that the package to which the class belongs is in the relation with the schema to which the table belongs. The relation AttributeToColumn is called from the where clause of ClassToTable and maps primitive attributes of the persistent class to columns of the corresponding table. The transformation also defines keys on metamodel elements, i.e. a definition of which properties of an element, in combination, can uniquely identify an instance of that class. In this case there is a key stating that the transformation must ensure that there cannot be two Tables with the same name within the same Schema. Below we show an excerpt of this transformation.

```
transformation uml2rdbms ( uml : UML , rdbms : RDBMS ) {
    key RDBMS::Table {name, schema};

    top relation PackageToSchema {
        pn : String;
        checkonly domain uml p : UML::Package { name = pn }; 
        enforce domain rdbms s : RDBMS::Schema { name = pn }; 
    }

    top relation ClassToTable {
        cn, prefix : String;
        checkonly domain uml c : UML::Class {
            namespace = p : UML::Package {}, kind = 'Persistent', name = cn 
        };
        enforce domain rdbms t : RDBMS::Table {
            schema = s : RDBMS::Schema {}, name = cn,
            column = cl : RDBMS::Column { name = ‘TID’, typeT = ‘NUMBER’ },
            keyK = k : RDBMS::Key { name = ‘PK’, column = cl } 
        };
        when { PackageToSchema(p, s); }
        where { AttributeToColumn(c, t, prefix); prefix = ''; }
    }

    relation AttributeToColumn { ... }
}
```

We introduce an institution $I^Q$ for QVT-Relations transformations (called QVTR) following the schema shown in Figure 3. Since transformations involve SW-models and metamodels, we base this institution on the $I^M$ institution defined in Section 3. The source and target metamodels that are involved in a specific model transformation are represented within the signature, and the source and target SW-models are represented in the model. Formulas represent the two basic conditions which must hold in a model transformation: keys defined on source and target metamodel elements, and transformation rules stating relations between source and target elements. The satisfaction relation checks if the keys hold in the corresponding SW-models, and it answers the following question: does the source and target SW-models satisfy the relations defined by the transformation rules?

Without losing potential, we restrict some of the QVT-Relations constructions [6]. 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 auxiliary functions and queries since they are syntactic sugar already supported with the basic constructions. Finally, we simplify the pattern structure by not considering opposite roles in object templates, since they can be expressed as conditions within a template, and collection templates, since they are advanced features not commonly used in practice.

We also forbid cycles of rule invocations to avoid infinite recursion. Notice that when and where clauses conform a potentially cyclic graph of dependencies between transformation rules. Cycles are however not problematic unless the satisfaction of a rule involving a set of SW-model elements depends recursively on its own satisfaction. In this case we have infinite recursion which cannot be handled by our proposal. We thus assume that recursion is well-founded, i.e. no rule will be called twice in the same chain of dependencies for the same set of elements. This constraint ensures
well-foundness since we always have a finite set of elements in any SW-model. Another alternative evaluated in [31] is to forbid cycles of dependencies to avoid infinite recursion. However, this alternative is too restrictive in practice.

In what follows we present how the elements of the institution are defined.

4.1. Signatures and Formulas

A signature defines the source and target metamodels that are involved in a specific model transformation.

**Definition 11 (QVTR signature).** A QVTR signature is a pair \( \langle \Sigma_{M1}, \Sigma_{M2} \rangle \) of \( I_M \)-signatures \( \Sigma_{Mi} = (C_i, \alpha_i, P_i) \) \((i = 1, 2)\) representing the source and target metamodels of the transformation.

We can either assume that there are no name clashes (types, roles and properties) between \( I_M \)-signatures or that equal names do not introduce an inconsistency. For example, in the case of name clash between type names, we assume that they are of the same type (e.g. for String). Moreover, there is no problem if there exist two properties with the same name and roles but with different parameters. If a transformation has the same source and target metamodels, we can use a prefix to identify elements on each side.

From this signature, we can derive a \( I_E \)-signature of the expressions language which must contain an element for each type \( \bigcup_i T(C_i) \) and a predicate for each property declaration \( \bigcup_i P_i \) in the \( I_M \)-signatures, as well as predefined predicates and functions for type constants and type constructors.

As depicted in Figure 3, formulas represent key constraints defined on source and target metamodel elements and transformation rules. Keys define which properties of an element, in combination, can uniquely identify an instance of that element, and are of the form \( \text{key Class } \{ \text{prop1}, \ldots, \text{propN} \} \). The identifying properties can also refer to non-navigable opposite roles, e.g. \( \text{key Class } \{ \text{prop1, opposite(Class2.property)} \} \). Top-level relations must hold in any transformation execution, and non-top-level relations are required to hold only when they are referred from another relation. We can view a relation as having the following abstract structure [6] (we consider only a source and a target metamodel).

![Figure 3: A model transformation as an institution](image-url)
Every relation has a set \(<\var_set>\) of variables occurring in the relation, which are particularly used within the domains \(<\domain\text{pat}>\) and in the when clause \(<\when\text{cond}>\). Relations define source/target domain patterns \(<\domain\text{pat}>\). A pattern can be viewed as a graph of typed elements (which will be matched by objects) and relations (which will be matched by links), together with a predicate (boolean expression) which must hold. The predicate may refer to variables other than the pattern elements; these are the free variables of a pattern. A pattern can be represented as:

\[
\begin{align*}
e_1: \text{classname}_1, e_2: \text{classname}_2 \ldots e_n: \text{classname}_N \\
\text{pattern can be represented as:}
\end{align*}
\]

\[
\text{hold. The predicate may refer to variables other than the pattern elements; these are the free variables of a pattern. A}
\]

Relations can also contain when \(<\when\text{cond}>\) and where \(<\where\text{cond}>\) clauses. A when clause specifies the conditions under which the relationship holds, whilst the where clause specifies the condition that must be satisfied by all SW-model elements participating in the relation, and it may constrain any of the variables in the relation and its domains. 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 \(<\text{R}_\var_set>\). In this sense, top-level relations can be parametric when called from a when clause, whereas non-top-level relations are always parametric since they are called for given source and target domains elements.

**Definition 12 (QVTR formula).** Given a signature \((\Sigma^1, \Sigma^2)\) such that \(\Sigma^M = (C_i, \alpha_1, P_i)\) with \(C_i = (C_i, \leq_{C_i})\) and \(P_i = (P_i, P_i)\), \(\Sigma\)-formulas are defined as follows:

- A formula \(\varphi^K\) representing a key constraint of the form \(\langle c, \{r_1, \ldots, r_n\} \rangle (1 \leq n)\) with \(c \in C_i, (j = 1..n)\) a class in one of the metamodels, \(r_j \in R_i, (j = 1..n)\) roles defined in properties in which such class participates (having such role or at the opposite side of it), i.e. for each \(r_j\) there is a property \(\langle r_j : c_i, r_i ; c_j \rangle\) or \(\langle r_i : c_i, r_j ; c_j \rangle\) \(P_i\) such that \(c = c_i\) (the property is non-navigable from \(c\)) or \(c = c_j\) (\(r_j\) is navigable from \(c\)). Roles determine the elements within these properties that together can uniquely identify an instance of the class.

- A formula \(\varphi^R\) representing a set of interrelated transformation rules with variables \(X^* = (X^*)_s \in \cup_i \text{T}(C_i)\), is a finite set of tuples representing rules of the form \(\langle \text{top}, \text{VarSet}, \text{ParSet}, \text{Pattern}_i (i = 1, 2)\rangle, \text{when, where}\), where:
  - \(\text{top} \in \{\text{true, false}\}\) defines if the rule is a top-level relation or not
  - \(\text{VarSet} \subseteq X^*\) is the set of variables used within the rule
  - \(\text{ParSet} \subseteq \text{VarSet}\) representing the set of variables taken as parameters when the rule is called from another one (corresponding to the top pattern element in the source and target domains, and the primitive domains defined within the rule)
  - \(\text{Pattern}_i (i = 1, 2)\) are the source and target patterns, i.e. tuples \(\langle E_i, A_i, P_{r_i} \rangle\) such that \(E_i \subseteq (X^*)_c \in C_i\), is a set of class-indexed variables, \(A_i\) is a set of elements representing associations of the form \(\text{rel}(p, x, y)\) with \(p \in P_i\) and \(x, y \in E_i\), and \(P_{r_i}\) is a \(\text{TF}\)-formula over these elements. We denote by \(\text{ParVarSet}_i (i = 1, 2)\) the variables used in pattern \(k\) that do neither occur in the other domain nor in the when clause.
  - \(\text{when/where}\) are the when(where) clauses of the rule, respectively. A when clause is a pair \(\langle \text{when}_c, \text{when}_e\rangle\) such that \(\text{when}_c\) is a \(\text{TF}\)-formula with variables in \(\text{VarSet}\), and \(\text{when}_e\) is a set of pairs of transformation rules (formulas) and set of variables which are the parameters of the rules. We will denote by \(\text{WhenVarSet}_i\) the set of variables occurring in the when clause. Finally, a where clause is a pair \(\langle \text{where}_c, \text{where}_e\rangle\) such
that where, $c$ is a $T_E$-formula with variables in $\text{VarSet}$, and where, $c$ is a set of pairs of transformation rules and set of variables (as before). Only variables used in a where clause (as prefix in the example) are contained in $2. \text{VarSet}$.

In the running example, the key definition is represented by the following formulas: $\langle \text{Table}, \{\text{name}, \text{schema}\}\rangle$. There is also a formula representing the whole transformation with the following rules (it lacks $\text{AttributeToColumn}$ which is not shown in the example).

$\text{PackageToSchema} = \langle \text{top}, \text{VarSet}, \text{ParSet}, \text{Pattern}_i (i = 1, 2), \text{when, where}\rangle$

- $\text{top} = \text{true}$
- $\text{VarSet} = \{\text{pn}, \text{p}, \text{s}\}$ with $\text{pn} \in X^{\text{String}}, \text{p} \in X^{\text{Package}}$, and $\text{s} \in X^{\text{Schema}}$
- $\text{ParSet} = \{\text{p}, \text{s}\}$
- $\text{Pattern}_1 = (E_1, A_1, Pr_1)$ with $E_1 = \{\text{p}\}$, $A_1 = \emptyset$, and $Pr_1 = \text{name}(\text{p}, \text{pn})$ Remember that $\text{name}(\text{p}, \text{pn})$ is a property in the source metamodel, and thus a predicate in the $T_E$ signature
- $\text{Pattern}_2 = (E_2, A_2, Pr_2)$ with $E_2 = \{\text{s}\}$, $A_2 = \emptyset$, and $Pr_2 = \text{name}(\text{s}, \text{pn})$
- $\text{when} = (\emptyset, \emptyset)$
- $\text{where} = (\emptyset, \emptyset)$

$\text{ClassToTable} = \langle \text{top}, \text{VarSet}, \text{ParSet}, \text{Pattern}_i (i = 1, 2), \text{when, where}\rangle$

- $\text{top} = \text{true}$
- $\text{VarSet} = \{\text{cn}, \text{prefix}, \text{c}, \text{p}, \text{Persistent}, \text{t}, \text{s}, \text{cl}, \text{NUMBER}, \text{TID}, \text{k}, \text{PK}\}$ with $\text{c} \in X^{\text{Class}}, \text{p} \in X^{\text{Package}}, \text{t} \in X^{\text{Table}}, \text{s} \in X^{\text{Schema}}, \text{cl} \in X^{\text{Column}}, \text{k} \in X^{\text{Key}}$, and the others in $X^{\text{String}}$
- $\text{ParSet} = \{\text{c}, \text{t}\}$
- $\text{Pattern}_1 = (E_1, A_1, Pr_1)$ with $E_1 = \{\text{c}, \text{p}\}$ $A_1 = \{\text{rel}((\text{namespace} : \text{Package}, \text{elements} : \text{Classifier}), \text{p}, \text{c})\}$ $Pr_1 = \text{name}(\text{c}, \text{cn})$ AND kind($\text{c}, \text{Persistent}$)
- $\text{Pattern}_2 = (E_2, A_2, Pr_2)$ with $E_2 = \{\text{rel}(\text{schema} : \text{Schema}, \text{tables} : \text{Table}, \text{s}, \text{t}), \text{rel}((\text{owner} : \text{Table}, \text{column} : \text{Column}), \text{t}, \text{cl}), \text{rel}((\text{column} : \text{Column}, \text{key} : \text{Key}), \text{cl}, \text{k}), \text{rel}((\text{owner} : \text{Table}, \text{key} : \text{Key}), \text{t}, \text{k})\}$ $Pr_2 = \text{name}(\text{t}, \text{cn})$ AND name($\text{cl}, \text{TID}$) AND type($\text{cl}, \text{NUMBER}$) AND name($\text{k}, \text{PK}$)
- $\text{when} = (\emptyset, \{(\text{PackageToSchema}, \{\text{p}, \text{s}\})\})$
- $\text{where}_e = \text{prefix} = \text{EMPTY}$

Now we have to define signature morphisms. We also need to prove that signatures and signature morphisms define a category $\text{Sign}$, and also that there is a functor $\text{Sen}$ from this category to the category of sets of formulas and their translations. The complete proofs are given in [16].

**Definition 13 (QVTR signature morphism).** Given a signature $\langle \Sigma_1^M, \Sigma_2^M \rangle$, a signature morphism is defined as a tuple of signature morphisms of the corresponding institutions $\langle \sigma_1^M, \sigma_2^M \rangle$. The signature morphism $\sigma^E$ is derived from the morphisms defined for types and predicates in $\sigma_1^M$ and $\sigma_2^M$.

Given a set of variables $X_2 = (X_2^1)_s \in (\bigcup_i T_2((C;i)))$, we define a set $X_2|_\sigma$ as $X_1 = (X_1^i)_s \in (\bigcup_i T_1((C;i)))$ by $X_1^s = X_2^s|_\sigma$. Signature morphisms extend to formulas over $\Sigma_1$ and $X_2|_\sigma$ as follows. Given a $\Sigma_1$-formula $\varphi$, $\sigma(\varphi)$ is the canonical application of the signature morphism to every element in $\varphi$. 

54
4.2. Models

As depicted in Figure 3, an interpretation contains a semantic representation for the source and target SW-models.

Definition 14 (QVTR interpretation). Given a signature $\langle \Sigma_1^M, \Sigma_2^M \rangle$, an interpretation is a tuple $\langle M_1^E, M_2^E \rangle$ of disjoint $\text{Sign}_1^M$-interpretations. Here we can also derive a $T^E$ model $M^E$ such that the interpretation of elements in $\text{Sign}_1^M$ must be the same in $M_1^M$ and $M_2^E$.

Back to the example, assume that we have an interpretation $M = \langle M_1^M, M_2^M \rangle$ such that $M_1^M$ is the one defined in Section 3.1, and $M_2^M$ is an interpretation with a direct correspondence with the SW-model in Figure 1d. The interpretation of elements in $\text{Sign}_1^M$ must be the same in $M_1^M$ and $M_2^E$.

Definition 15 (Binding of variables). Given a signature $\langle \Sigma_1^M, \Sigma_2^M \rangle$ such that $\Sigma_1^M = \langle C_i, a_i, P_i \rangle$ with $C_i = \langle C_i, \leq C_i, \rangle$ and $P_i = \langle R_i, P_i \rangle$, an interpretation $\langle M_1^M, M_2^M \rangle$, and variables $X^s = \langle X^s, x \in (U \setminus T(C)) \rangle$, the binding of a variable $x^c \in X^c$, denoted by $|x^c|$, is the set of any possible interpretation of such variable, which corresponds to $|x^c| = V_c$ if $c$ is a class, or corresponds to this set together with the elements created using type constructors (in $M^E$) in the case $c$ is a primitive type. Moreover, the binding of a set of variables $(x_1, \ldots, x_n)$, denoted by $|(x_1, \ldots, x_n)|$, is defined as $\{(y_1, \ldots, y_n) \mid y_i \in |x_i| (i = 1..n)\}$. We can also view $|(x_1, \ldots, x_n)|$ as a set of variable assignments. We denote by $\mu[x_1, \ldots, x_n]$ a function with an assignment for variables $x_1, \ldots, x_n$. 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 $\mu_2$.

Binding of variables depends on the type of elements. For a class variable, we have that the set of possible values coincides with the set of elements within the CSMOF institutions. For example, we have that $|p| = V_{\text{Package}} = \{p\}$. However, if the variable is of a primitive type, since transformation rules can use other elements besides those in the CSMOF institutions (for example these strings created using the type constructor $+$), we can have more elements. In the example, we have that $|\mu| = \{\text{Pac}, \text{Str}, \text{ID}, \ldots, \text{numb}, \ldots, \text{ID} + \text{tid}, \text{ID} + \text{numb}, \ldots\}$.

Definition 16 (QVTR homomorphism and reduct). Given signatures $\Sigma_i = \langle \Sigma_1^M, \Sigma_2^M \rangle (i = 1, 2)$, a signature morphism $\sigma : \Sigma_1 \rightarrow \Sigma_2$, and $\Sigma_2$-interpretation $M = \langle M_1^M, M_2^M \rangle$ and $M_2 = \langle M_1^M, M_2^M \rangle$, homomorphisms and reducts are defined componentwise. A $\Sigma_2$-homomorphism $h : M \rightarrow M_2$ is defined as a tuple of homomorphisms $\langle h_1^M, h_2^M \rangle$ of the corresponding institutions. The reduct $M_1^M_{\sigma}$ of $M$ along $\sigma$ is the $\Sigma_1$-interpretation $\langle M_1^M|_{\sigma}, M_2^M|_{\sigma} \rangle$. Moreover, the reduct $h|_\sigma$ of $h$ along $\sigma$ is the $\Sigma_1$-homomorphism $\langle h_1^M|_{\sigma}, h_2^M|_{\sigma} \rangle$.

We can prove that interpretations and homomorphisms define a category $\text{Mod}(\Sigma)$. We can also prove that the reduct defines a functor and thus there is a functor $\text{Mod}$ giving a category of interpretations for each signature and a functor defined by the reduct. The complete proofs are given in [16].

4.3. Satisfaction Relation and Satisfaction Condition

The execution of a transformation requires that all its top-level relations hold, as well as the keys (if any). However the effect of a transformation depends on the direction of its execution, and on the flags that can be attached to domains. Domains can be defined as checkonly or enforced. As the standard says “When a transformation is enforced in the direction of a checkonly domain, it is simply checked to see if there exists a valid match in the relevant model that satisfies the relationship. When a transformation executes in the direction of the model of an enforced domain, if checking fails, the target model is modified so as to satisfy the relationship.” [6]. These two execution modes are determined by two different semantics of a rule: the enforcement and checking semantics. The first determines how a model transformation builds a target SW-model, in accordance with the general schema of model transformations presented in Section 1. The second, more useful for verification, determining what relations must exist between source and target SW-models, without previous knowledge about if the target model was built executing the transformation in enforcement mode or not.

Since we are interested in defining MDE elements and their relations, the satisfaction relation of our institution is defined in terms of the standard checking semantics, i.e. it checks whether the source and target SW-models (represented within the interpretation) satisfy the relations defined by the transformation rules. It also checks whether key constraints hold (both represented as formulas).
Without loss of expressiveness, we consider only one source and target metamodels and we also consider that the transformation is executed in the direction of the second domain. In this context, we customize the standard checking semantics and take the first and second patterns as the source and target patterns, respectively. Using $<\text{var}_\text{set}>$ as a binding of variables of the set $\text{var}_\text{set}$, and $\text{exec}_\text{domain}_k\text{var}_\text{set}$ as the variables of domain $k$ that do neither occur in the other domain nor the when clause, the standard states that a rule holds if:

$$\forall\langle<\text{when}_\text{var}_\text{set}>\rangle, (\langle<\text{when}_\text{cond}>\rightarrow$$

$$\forall\langle<R\text{var}_\text{set}>\langle<\text{when}_\text{var}_\text{set}>\cup\langle<\text{exec}_\text{domain}_2\text{var}_\text{set}>\rangle\rangle, (\langle<\text{domain}_1\text{pat}>\rightarrow$$

$$\exists\langle<\text{exec}_\text{domain}_2\text{var}_\text{set}>\rangle, (\langle<\text{domain}_2\text{pat}>\wedge<\text{where}_\text{cond}>\rangle))$$

This formula 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.

**Definition 17 (QVT satisfaction relation).** Given a signature $(\Sigma_1^\mathit{M}, \Sigma_2^\mathit{M})$ such that $\Sigma_1^\mathit{M} = (C_i, \alpha_i, P_i)$ with $C_i = (C_{i1}, \leq_{C_i})$ and $P_i = (R_{i1}, P_{i1})$, and an interpretation $\mathcal{M} = (\mathcal{M}_1^\mathit{M}, \mathcal{M}_2^\mathit{M})$, we define that $\mathcal{M}$ satisfies a formula $\varphi$, written $\mathcal{M} \models \varphi$, in one of the following cases:

- A formula $\varphi^\mathit{K} = \langle c, (r_1, \ldots, r_m) \rangle$ with $c \in C_i$ (j = 1..n), $r_j \in R_i$ (j = 1..n), is satisfied in the corresponding metamodel $\mathcal{M}_j^\mathit{M}$ if there are no two elements of type $c$ with the same set of elements related through properties involving roles $r_j$ (they must differ in at least one element). Formally, for each $r_j$ the corresponding property $p_j$ is $(\langle r : c, r_j : d \rangle)$ if $r_j$ is navigable, or $(\langle r_j : c : d \rangle)$ if the opposite role of $r_j$ is non-navigable. We can define that given an element $x \in (V_c)_{c \in C_i}$, the set of semantic elements linked with $x$ in $p_j$ is $\nu(x, p_j) = \{\pi_2(t) | \pi_1(t) = x, t \in p_j^\mathit{M}\}$. For all $x, y \in (V_c)_{c \in C_i}, x \neq y$ implies $\bigcup_j \nu(x, p_j) \neq \bigcup_j \nu(y, p_j)$.

- A formula $\varphi^\mathit{R}$ is satisfied if the semantics defined in the standard [6] holds, i.e. if every top-level relation holds, which means that there are matching elements in the source and target SW-models in the relation. Formally, given a $\mathcal{I}_\mathit{E}$ model $\mathcal{M}_\mathit{E}$ built from the interpretation $\mathcal{M}$, $\varphi^\mathit{R}$ is satisfied if for every top rule $\text{Rule} \in \varphi^\mathit{R}$, we have that $\mathcal{M}_\mathit{E}, \emptyset \models \text{Rule}$. We use $\emptyset$ as the empty variable assignment, only filled in the case of explicit called rules.

A rule $\text{Rule} = \langle\text{top}_\mathit{Var}\text{Set}, \text{Par}\text{Set}, \text{Pattern}_i (i = 1, 2), \text{when}, \text{where}\rangle$ is satisfied with respect to a model $\mathcal{M}_\mathit{E}$ and a variable assignment $\mu$, denoted by $\mathcal{M}_\mathit{E}, \mu \models \text{Rule}$ if:

1. If $\text{When}\text{Var}\text{Set} = \emptyset$

   $$\forall \mu^1[x_1, \ldots, x_n] \in \text{Var}\text{Set}\setminus\text{2}\text{Var}\text{Set}, (\mathcal{M}_\mathit{E}, (\mu^1[x_1, \ldots, x_n] \cup \mu) \models \text{Pattern}_1 \rightarrow$$

   $$\exists \mu^2[y_1, \ldots, y_m] \in \text{2}\text{Var}\text{Set}, (\mathcal{M}_\mathit{E}, (\mu^1 \cup \mu^2 \cup \mu) \models \text{Pattern}_2 \wedge \mathcal{M}_\mathit{E}, (\mu^1 \cup \mu^2 \cup \mu) \models \text{where}))$$

2. If $\text{When}\text{Var}\text{Set} \neq \emptyset$

   $$\forall \mu^w[z_1, \ldots, z_0] \in \text{When}\text{Var}\text{Set}, (\mathcal{M}_\mathit{E}, (\mu^w[z_1, \ldots, z_0] \cup \mu) \models \text{when} \rightarrow$$

   $$\forall \mu^1[x_1, \ldots, x_n] \in \text{Var}\text{Set}\setminus(\text{When}\text{Var}\text{Set} \cup \text{2}\text{Var}\text{Set}), (\mathcal{M}_\mathit{E}, (\mu^1 \cup \mu^w \cup \mu) \models \text{Pattern}_1 \rightarrow$$

   $$\exists \mu^2[y_1, \ldots, y_m] \in \text{2}\text{Var}\text{Set}, (\mathcal{M}_\mathit{E}, (\mu^1 \cup \mu^2 \cup \mu^w \cup \mu) \models \text{Pattern}_2 \wedge$$

   $$\mathcal{M}_\mathit{E}, (\mu^1 \cup \mu^2 \cup \mu^w \cup \mu) \models \text{where}))$$

A pattern $\text{Pattern} = \langle E_A, P_r \rangle$ is satisfied with respect to a model $\mathcal{M}_\mathit{E}$ and a variable assignment $\mu$ (which must include a valuation for elements in $E$), denoted by $\mathcal{M}_\mathit{E}, \mu \models \text{Pattern}$ if there is a matching subgraph of elements and the predicate holds, i.e.

- $\forall \text{rel}((r_1 : c_1, r_2 : c_2), x, y) \in A. (\mu(x), \mu(y)) \in \mathcal{M}_\mathit{E}$ (this means that the model $\mathcal{M}_\mathit{E}$ has a relation (corresponding to the property $\langle r_1 : c_1, r_2 : c_2 \rangle$ connecting elements $\mu(x)$ and $\mu(y)$)
\[ M^E, \mu \models_E \phi \] such that \( \models_E \) is the satisfaction relation in \( T^E \)

A \( \text{when} \) clause \( \langle \text{when}_c, \text{when}_r \rangle \) is satisfied with respect to a model \( M^E \) and a variable assignment \( \mu \), denoted by \( M^E, \mu \models \langle \text{when}_c, \text{when}_r \rangle \) if

\[ M^E, \mu \models_E \text{when}_c \land (\forall (r, v) \in \text{when}_r. \ M^E, \mu [v] \models r) \]

such that \( \models_E \) is the satisfaction relation in \( T^E \), and the later is the satisfaction of the parametric transformation rule \( r \) (which must be defined in the formula \( \varphi^E \)) using the variable assignment \( \mu [v] \) as a parameter. The satisfaction of a \text{where} clause is defined in the same way.

In the example, we need to prove that our interpretation satisfies both kinds of formulas. In the case of the key \( \langle \text{Table}, \{ \text{name}, \text{schema} \} \rangle \), we have only one table \( t \) and only one key \( k \), thus the condition trivially holds. We need to prove now that \( M^E, \emptyset \models \text{PackageToSchema} \).

We know that \( \{ \text{pn} \} = \{ \text{Pac, Str, ID, Per, val, numb, numb, varch, ...} \} \), and also that \( \{ \text{p} \} = V_{\text{package}} = \{ \text{p1} \} \), thus \( \{ \text{pn, p} \} \) is \( \{ \text{Pac, p1}, (\text{Str, p1}), (\text{ID, p1}), ... \} \). We also have that \( s = V_{\text{schema}} = \{ \text{s1} \} \). Thus, \( M^E, \emptyset \models \text{PackageToSchema} \) if

\[
\forall \mu^1[\text{pn}, \text{p}] \in \{(\text{Pac, p1}), (\text{Str, p1}), (\text{ID, p1}), (\text{Per, p1}), (\text{val, p1}), (\text{numb, p1}), ...\},
\quad (M^E[\varphi], \mu^1 \models \text{Pattern}_1 \rightarrow \exists \mu^2[s] \in \{ \{s1\} \}, (M^E[\varphi], \mu^1 \cup \mu^2 \models \text{Pattern}_2 \land M^E[\varphi], \mu^1 \cup \mu^2 \models \text{where}))
\]

For every \( \mu^1[\text{pn}, \text{p}] \) different from \( \{ \text{Pac, p1} \} \) we have that \( \text{Pattern}_1 \) does not hold, since it depends on the predicate \( \text{name}(\text{p}, \text{pn}) \). Thus, in these cases the implication holds. Now, in the case of \( \{ \text{Pac, p1} \} \), we have that \( \text{Pattern}_1 \) holds, and that the only possible value for \( s \) is \( s1 \). In this case, we also have that \( M^E, (\mu^1 \cup \mu^2) \models \text{Pattern}_2 \) since the predicate \( \text{name} (s, \text{pn}) \) holds. Note at the relational model in Figure 1d that the schema has the same name as the package, which is semantically represented as \( \text{Pac} \). Moreover, since the \text{where} clause is empty, \( M^E, (\mu^1 \cup \mu^2) \models \text{where} \) trivially holds. In conclusion, we have that \( M^E, \emptyset \models \text{PackageToSchema} \).

Finally, we need to prove that \( M^E, \emptyset \models \text{ClassToTable} \). Proceeding in the same way, we have to prove that:

\[
\forall \mu^w[\text{p}, \text{s}] \in \{(\text{p1, s1})\}, (M^E, \mu^w[\text{p}, \text{s}] \models \text{when} \rightarrow \forall \mu^1 \in \{(c, c, \text{Persistent})\}, (M^E, (\mu^1 \cup \mu^w) \models \text{Pattern}_1 \rightarrow
\exists \mu^2 \in \{(\text{prefix}, \text{t, cl, NUMBER, TID, k, Per})\}, (M^E, (\mu^1 \cup \mu^2 \cup \mu^w) \models \text{Pattern}_2 \land M^E, (\mu^1 \cup \mu^2 \cup \mu^w) \models \text{where}))
\]

We have a \text{when} clause which is the invocation of the relation \text{PackageToSchema} with a concrete variable assignment for domain variables \( \text{p} \) and \( \text{s} \). We have proved above that with this assignment \( M^E, \mu^w[\text{p}, \text{s}] \models \text{PackageToSchema} \) holds. Now, in the case of \( M^E, (\mu^1 \cup \mu^w) \models \text{Pattern}_1 \) we need to prove that \( \mu^1(\text{p}), \mu(c) \in M^E \) since

\[
\text{rel}((\text{namespace} : \text{Package}, \text{elements} : \text{Classifier}), \text{p}, c) \in A. \quad \text{We also need to prove that} \quad M^E, (\mu^1 \cup \mu^w) \models_E P_{r1} \quad \text{with} \quad P_{r1} = \text{name}(c, c) \text{ AND kind}(c, \text{Persistent}). \quad \text{This only holds with the variable assignment} \quad \mu^1[c, c, \text{Persistent}] = (c, \text{ID}, \text{Per}) \text{ and } \mu^w[p] = p1. \quad \text{In any other case, Pattern}_1 \text{ does not hold and thus the rest of the expression holds.}
\]

Now, for proving \( M^E, (\mu^1 \cup \mu^2) \models \text{Pattern}_2 \) we need to prove that

- \( (\mu(s), \mu(t)) \in \text{E} \text{since } \text{rel}((\text{schema} : \text{Schema}, \text{tables} : \text{Table}), t, s) \in A \)
- \( (\mu(t), (\mu(cl)) \in \text{E} \text{since } \text{rel}((\text{owner} : \text{Table}, \text{column} : \text{Column}), t, cl) \in A \)
- \( (\mu(cl), \mu(k)) \in \text{E} \text{since } \text{rel}((\text{column} : \text{Column}, \text{key} : \text{Key}), cl, k) \in A \)
- \( (\mu(t), (\mu(k)) \in \text{E} \text{since } \text{rel}((\text{owner} : \text{Table}, \text{key} : \text{Key}), t, k) \in A \)

and also that \( M^E, (\mu^1 \cup \mu^2) \models_E P_{r2} \). These expressions hold with \( \mu^2[\text{prefix}, t, \text{cl, NUMBER, TID, k, PK}] = (\text{null}, t, c1, \text{num}, \text{TID}, k1, \text{PK}) \). Finally, with the variable assignment we have at the moment \( (\mu^1 \cup \mu^2 \cup \mu^w)[... ] = (\text{ID}, c1, p1, \text{Per, null, t1, s1, cl1, num, TID, k1, PK}) \), we can prove \( M^E, \mu^w[t, \text{prefix}] \models \text{AttributeToColumn} \).

We close the definition of the institution by proving the satisfaction condition. No proof assistant was used.
Theorem 1 (QVTR satisfaction condition). Given signatures $\Sigma_i$ ($i = 1, 2$), a signature morphism $\sigma : \Sigma_1 \rightarrow \Sigma_2$, a $\Sigma_2$-interpretation $M = (M^1, M^2)$, a set of variables $X_2 = (X^2_2)_s \in \Sigma_2$, and a $\Sigma_1$-formula $\varphi$ with variables in $X_2|\sigma$, the following satisfaction condition holds.

$$M|_\sigma \models \Sigma_1 \varphi \quad \text{iff} \quad M \models \Sigma_2 \sigma(\varphi).$$

**Proof (Proof Sketch).** In the case of a formula $\varphi^M = \langle c, \{r_1, ..., r_n\} \rangle$ with $c \in C_i$ ($j = 1..n$), $r_j \in R_i$ ($j = 1..n$), it is satisfied in the corresponding interpretation $M_i^M|_\sigma$ if there are no two elements of type $c$ with the same set of elements related through properties involving roles $r_j$ (they must differ in at least one element). Since the signature morphism changes types and roles consistently, the interpretation $M_i^M$ will not have more or less elements of type $\sigma_T(c)$, or less or more elements related through properties involving roles $\sigma_R(r_j)$. Thus, the translated formula is also satisfied. This implication also holds backwards. In the case of a formula $\varphi^R$, the reasoning is similar. The formula holds if every top-level relation holds, which means that there are matching elements in the source and target interpretations within $M_i|_\sigma$ satisfying the relations. After the signature morphism, we can find the same matching elements in the source and target interpretations within $M$, and since the signature morphism can only introduce new types and roles not used within $\sigma(\varphi^R)$, the relations between matching elements still hold. Once again, this implication also holds backwards. Finally, the satisfaction condition holds.

Given that the satisfaction condition holds we can state that $T^\Omega$ consisting of signatures, morphisms, formulas, interpretation, reducts, homomorphisms, and the satisfaction relation, defines an institution.

5. Extending the Institutions

From a proof-theoretic point of view, we need to use an entailment system such that it is possible to prove that multiplicity constraints (or other constraints) hold in a SW-model, which is the context in which the verification must be done. For this we need to extend the definition of CSMOF formulas. The extension allows the inclusion of SW-models as syntactic elements (represented as a formula $\Omega$) to be considered by any possible entailment system devised to derive the satisfiability of other formulas, represented as a set of formulas $\Psi$, i.e. $\Omega \vdash_\Sigma \Psi$. In the same way, we also need to verify whether a key constraint (or a set of them), represented as a formula $\varphi^K$, is derived from the same $\Omega$, i.e. $\Omega \vdash_\Sigma \varphi^K$, or whether a transformation rule $\varphi^R$ (or the whole model transformation) is derived from a pair of SW-models, i.e. $\Omega_1 \cup \Omega_2 \vdash_\Sigma \varphi^R$. A sound entailment system will ensure semantic entailment, i.e. $\Omega \vdash_\Sigma \Psi$ implies $\Omega \models_\Sigma \Psi$. Semantic entailment is defined by the satisfaction relations of the institutions.

5.1. An Institution for SW-Models

We first define a supporting institution $T^\Omega$ by taking the same basic definitions from the CSMOF institution given in Section 3 and changing the definition of institutions, as well as the corresponding satisfaction relation. A formula in this institution is a syntactic representation of a SW-model.

**Definition 18 (T^\Omega-formulas).** Given a signature $\Sigma = (C, \alpha, P)$ with $C = (C, \subseteq C)$ and $P = (R, P)$ as defined for the CSMOF institution, and variables $X = (X^c)_{c \in T(C)}$, we define formulas as follows:

$$\Omega :::= \ x^c \ | \ \{r_1, x_1^{c_1}, r_2, x_2^{c_2}\} \ | \ \Omega \oplus \Omega$$

with $x^c \in X^c$, $x_1^{c_1} \in X^{c_1}$, $x_2^{c_2} \in X^{c_2}$, $c, c_1, c_2 \in T(C)$, $\{r_1 : c_1, r_2 : c_2\} \in P$, $r_1, r_2 \in R$.

A variable $x^c$ represents a typed element, $\{r_1, x_1^{c_1}, r_2, x_2^{c_2}\}$ represents a link between two typed elements with their respective roles, and $\Omega \oplus \Omega$ allows to compose these elements to represent a whole SW-model.

The extension of a signature morphism $\sigma$ to a formula $\Omega$ is still the canonical application of the signature morphism to every type and role in the formula. Since signature and signature morphisms are taken from the CSMOF institution, the category $\text{Sign}$ is still defined. We can also prove that we still have a functor $\text{Sen}$ giving a set of formulas for each signature and a function translating sentences for each signature morphism. The complete proofs are given in [16].

We need to define a reduction of an interpretation with respect to the types used in the definition of the SW-model formula $\Omega$. This reduction defines an explicit scope in which the satisfaction of a formula is checked. We first define the types used within a formula as follows.
Definition 19 (Types of a $\mathcal{T}$-formula). Given a signature $\Sigma = (C, \alpha, P)$ with $C = (C, \leq_C)$ and $P = (R, P)$, variables $(X_c)_{c \in T(C)}$, and a $\Sigma$-formula $\Omega$ representing a SW-model, the function types giving the set of types used within $\Omega$ is inductively defined as follows:

- types$(x^c) = \{c\}$
- types$((r_1, x_1^{c_1}, r_2, x_2^{c_2})) = \{c_1, c_2\}$
- types$(\Omega_1 \oplus \Omega_2) = \text{types}(\Omega_1) \cup \text{types}(\Omega_2)$

Definition 20 (Explicit scope). Given a signature $\Sigma = (C, \alpha, P)$ with $C = (C, \leq_C)$ and $P = (R, P)$, a $\Sigma$-formula $\Omega$ representing a SW-model, and a $\Sigma$-interpretation $\mathcal{I} = (V^I_{\mathcal{I}}(C), A)$, the explicit scope defined by $\Omega$ in $\mathcal{I}$, denoted by $\mathcal{I}[\Omega]$, is the $\Sigma$-interpretation $(V^I_{\mathcal{I}}(\Omega), A[\Omega])$ such that:

- $V^I_{\mathcal{I}}(\Omega)|_{\mathcal{I}} = (V_c, c \in \text{types}(\Omega))$ with $V_c \in V^I_{\mathcal{I}}(C)$
- $A[\Omega]$ only contains those relations $\langle r_1 : c_1, r_2 : c_2 \rangle^\mathcal{I} \in A$ such that $c_1, c_2 \in \text{types}(\Omega)$

We define the satisfaction relation for $\Omega$ formulas based on a valuation function $K^\mathcal{I}$ determining that there is an isomorphism between $\Omega$ and the interpretation $\mathcal{I}$.

Definition 21 (Value of a $\mathcal{T}$-$\Omega$-formula). Given a signature $\Sigma = (C, \alpha, P)$ with $C = (C, \leq_C)$ and $P = (R, P)$, variables $(X_c)_{c \in T(C)}$, a $\Sigma$-formula $\Omega$ representing a SW-model, and a $\Sigma$-interpretation $\mathcal{I} = (V^I_{\mathcal{I}}(C), A)$, we define a valuation $K^\mathcal{I}(\Omega)$ as a bijective function mapping each $x^c \in \omega(\Omega)$ to an element of $(V_c, c \in T(C))$ for each $c \in T(C)$, such that syntactic links in $\Omega$ and semantic links in $\mathcal{I}$ coincide, i.e.

- for every $\langle r_1 : c, r_2 : d \rangle^\mathcal{I} \in A$, $\langle r_1 : c, r_2 : d \rangle^\mathcal{I} \in P$
- for every $\langle r_1 : c, r_2 : d \rangle^\mathcal{I} \in A$, $\langle r_1 : c, r_2 : d \rangle^\mathcal{I} = \{(K^\mathcal{I}(x^c), K^\mathcal{I}(y^d)) | \langle r_1 : c, r_2 : d \rangle^\mathcal{I} \in \omega(\Omega)\}$

The function $\nu$ gives the set of variables within a formula $\Omega$, i.e.

- $\nu(x^c) = \{x^c\}$
- $\nu(\langle r_1 : c_1, r_2, x_2^{c_2} \rangle) = \{x_1^{c_1}, x_2^{c_2}\}$
- $\nu(\Omega_1 \oplus \Omega_2) = \nu(\Omega_1) \cup \nu(\Omega_2)$

The function $\omega$ gives the set of links within a formula $\Omega$, i.e.

- $\omega(x^c) = \emptyset$
- $\omega(\langle r_1 : c_1, r_2, x_2^{c_2} \rangle) = \{\langle r_1 : c_1, r_2, x_2^{c_2} \rangle\}$
- $\omega(\Omega_1 \oplus \Omega_2) = \omega(\Omega_1) \cup \omega(\Omega_2)$

Finally, we can now define the satisfaction relation of a $\Omega$ formula with respect to an interpretation $\mathcal{I}$ as the existence of the isomorphic function between the SW-model and the explicit scope defined by the reduced interpretation with respect to the types in the SW-model.

Definition 22 ($\mathcal{T}$-$\Omega$ satisfaction relation). Given a signature $\Sigma$, a $\Sigma$-formula $\Omega$ representing a SW-model, and a $\Sigma$-interpretation $\mathcal{I}$, the satisfaction relation is defined as follows:

$\mathcal{I} \models^\Sigma \Omega$ if there exists a valuation function $K^\mathcal{I}(\Omega)$

The definitions of interpretations, reducts and homomorphisms do not change, thus original properties of the CSMO$\Omega$ institution with respect to these elements still hold: the existence of the category Mod$\Sigma$, the functorial properties of reduct, and the existence of the functor Mod. Nevertheless, we need to prove the satisfaction condition, as shown in the following theorem.
Theorem 2 (\(I^\Omega\) satisfaction condition). Given signatures \(\Sigma_i\) \((i = 1, 2)\), a signature morphism \(\sigma : \Sigma_1 \to \Sigma_2\), a \(\Sigma_2\)-interpretation \(\mathcal{I}\), and a \(\Sigma_1\)-formula \(\Omega\), the following satisfaction condition holds.

\[
\mathcal{I}|_\sigma \models_{\Sigma_1} \Omega \iff \mathcal{I} \models_{\Sigma_2} \sigma(\Omega)
\]

Proof (Proof Sketch). We know by definition that \(\mathcal{I}|_\sigma \models_{\Sigma_1} \Omega\) holds if there exists a bijective function \(K^{(I_\Omega)}|_{\sigma}(\Omega)\), i.e. a bijective mapping variables in \(\Omega\) such that syntactic links in \(\Omega\) and semantic links in \((\mathcal{I}|_{\sigma(\Omega)})\) coincide. Since this interpretation is restricted to those types found in the formula \(\Omega\), we can find the same elements (no more or less) in \(\mathcal{I}|_{\sigma(\Omega)}\). We can have more elements in \(\mathcal{I}\) giving an interpretation to types and properties introduced by the signature morphism. However, these types and properties will not be considered in the translated formula \(\sigma(\Omega)\) and thus they will not be part of \(\mathcal{I}|_{\sigma(\Omega)}\). In conclusion, there is also a bijective function \(K^{(I_\Omega)}|_{\sigma(\Omega)}(\sigma(\Omega))\). This implication also holds backwards, and thus, the satisfaction condition holds.

Given that the satisfaction condition holds we can state that \(I^\Omega\) consisting of signatures, morphisms, formulas, interpretations, reducts, and the satisfaction relation, defines an institution.

5.2. Extending CSMOF and QVTR

We can easily combine this supporting institution \(I^\Omega\) and the original CSMOF and QVTR institutions, since the only differences between them are the definition of formulas and the corresponding satisfaction relation. In this context, we can define an extended CSMOF institution \(I^{M^+}\) in which formulas are the disjoint union of both kind of formulas.

Definition 23 (Extended CSMOF formulas). Given a CSMOF signature \(\Sigma\), an extended CSMOF formula is defined as the disjoint union of CSMOF formulas and \(I^\Omega\) formulas, i.e.

\[
\Psi ::= \Phi | \Omega
\]

with \(\Phi\) a CSMOF formula defined in Section 3.1, and \(\Omega\) a \(I^\Omega\) formula defined in Section 5.1.

Since the definitions of signature and signature morphisms coincide in both institutions, the category Sign is still defined. Moreover, we have that the functor Sen is defined for CSMOF formulas, and for \(I^\Omega\) formulas. By using the fact that a \(I^{M^+}\)-formula is the disjoint union of both kind of formulas, we can straightforwardly conclude that the functor Sen is still defined in this extended CSMOF institution.

Finally, we need to state the satisfaction relation.

Definition 24 (Extended CSMOF satisfaction relation). Given a signature \(\Sigma\), a \(\Sigma\)-formula and a \(\Sigma\)-interpretation \(\mathcal{I}\), the interpretation satisfies the formula in one of the following cases:

- \(\mathcal{I} \models_{\Sigma} \Phi\) as in Definition 10
- \(\mathcal{I} \models_{\Sigma} \Omega\) as in Definition 22

As with the institution \(I^\Omega\), the definitions of interpretations, reducts and homomorphisms do not change, thus original properties of the CSMOF institution with respect to these elements still hold. Moreover, the satisfaction condition holds.

Theorem 3 (\(I^{M^+}\) satisfaction condition). Given signatures \(\Sigma_i\) \((i = 1, 2)\), a signature morphism \(\sigma : \Sigma_1 \to \Sigma_2\), a \(\Sigma_2\)-interpretation \(\mathcal{I}\), and a \(\Sigma_1\)-formula \(\varphi\), the following satisfaction condition holds.

\[
\mathcal{I}|_\sigma \models_{\Sigma_1} \varphi \iff \mathcal{I} \models_{\Sigma_2} \sigma(\varphi)
\]

Proof (Proof Sketch). In the case of \(\varphi\) a CSMOF formula, we have that the satisfaction condition holds. Moreover, in the case of \(\varphi\) a \(I^\Omega\) formula, we also have that the satisfaction condition holds, as proved by Theorem 2. By using the fact that a \(I^{M^+}\)-formula is the disjoint union of both kind of formulas, we can straightforwardly conclude that the satisfaction condition holds for the union of them.
Given that the satisfaction condition holds we can state that $\mathcal{I}^{M^+}$ defines an extended CSMOF institution.

The extension of QVTR is very similar. We can define an extended QVTR institution $\mathcal{I}^{Q+}$ in which formulas are the disjoint union of QVTR formulas and $\mathcal{I}^{M^+}$ formulas.

**Definition 25 (Extended QVTR formulas).** Given a QVTR signature $\Sigma$, an extended QVTR formula is defined as the disjoint union of QVTR formulas and extended $\mathcal{I}^{M^+}$ formulas, i.e.

$$\Pi := \Psi_{i} \mid \varphi_{K} \mid \varphi_{R}$$

with $\Psi_{i}$ ($i = \{1, 2\}$) a $\mathcal{I}^{M^+}$ formula, as defined before, indexed by the institution in which it is defined; $\varphi_{K}$ a key constraint, and $\varphi_{R}$ a set of interrelated transformation rules, both defined in Section 4.1.

Since the definitions of signature and signature morphisms coincide with the QVTR institutions, the category $\text{Sign}$ is still defined. Moreover, we have that the functor $\text{Sen}$ is defined for QVTR formulas, and in the last section we have the same for extended CSMOF formulas. By using the fact that a $\mathcal{I}^{Q+}$-formula is the disjoint union of both kind of formulas, we can straightforwardly conclude that the functor $\text{Sen}$ is still defined in this extended QVTR institution. Finally, we state the satisfaction relation.

**Definition 26 (Extended QVTR satisfaction relation).** Given a signature $\Sigma$, a $\Sigma$-formula and a $\Sigma$-interpretation $\mathcal{I} = \langle \mathcal{M}_{1}, \mathcal{M}_{2} \rangle$, the interpretation satisfies the formula in one of the following cases:

- $\langle \mathcal{M}_{1}, \mathcal{M}_{2} \rangle \models_{\Sigma} \Psi_{i}$ if $\mathcal{M}_{i} \models_{\Sigma} \Psi_{i}$ as in Definition 24.
- $\langle \mathcal{M}_{1}, \mathcal{M}_{2} \rangle \models_{\Sigma} \varphi_{K}$ as in Definition 17
- $\langle \mathcal{M}_{1}, \mathcal{M}_{2} \rangle \models_{\Sigma} \varphi_{R}$ as in Definition 17

The definitions of interpretations, reducts and homomorphisms is the same as with the QVTR institution, thus the properties with respect to these elements still hold. Moreover, the satisfaction condition holds.

**Theorem 4 ($\mathcal{I}^{Q+}$ satisfaction condition).** Given signatures $\Sigma_{i}$ ($i = 1, 2$), a signature morphism $\sigma : \Sigma_{1} \rightarrow \Sigma_{2}$, a $\Sigma_{2}$-interpretation $\mathcal{I}$, and a $\Sigma_{1}$-formula $\varphi$, the following satisfaction condition holds.

$$\mathcal{I}_{\sigma} \models_{\Sigma_{1}} \varphi \iff \mathcal{I} \models_{\Sigma_{2}} \sigma(\varphi)$$

**Proof (Proof Sketch).** In the case of $\varphi$ a QVTR formula, we have that the satisfaction condition holds, as proved by Theorem 1. Moreover, in the case of $\varphi$ an extended CSMOF formula, we also have that the satisfaction condition holds, as proved by Theorem 3. By using the fact that a $\mathcal{I}^{Q+}$-formula is the disjoint union of both kind of formulas, we can straightforwardly conclude that the satisfaction condition holds for the union of them.

Given that the satisfaction condition holds we can state that $\mathcal{I}^{Q+}$ defines an extended QVTR institution.

5.3. Running Example

The supporting institution allows the definition of a formula $\Omega$ corresponding to the class SW-model in Figure 1c as follows:

$$\text{p}^\text{Package} \oplus \text{c}^\text{Class} \oplus \text{a}^\text{Attribute} \oplus \text{pdt}^\text{PrimitiveDataType} \oplus \text{Package}^\text{String} \oplus \text{ID}^\text{String} \oplus \text{Persistent}^\text{String} \oplus ... \oplus \langle \text{namespace}, \text{p}, \text{elements}, \text{c} \rangle \oplus \langle \text{p}, \text{name}, \text{Package} \rangle \oplus \langle \text{namespace}, \text{p}, \text{elements}, \text{pdt} \rangle \oplus \langle \text{a}, \text{type}, \text{pdt} \rangle \oplus ...$$

The types used for the definition of the $\Sigma$-formula $\Omega$ coincide with those in the signature $\Sigma$ in Section 3.1. For such reason, the interpretation in the same section cannot be reduced with respect to $\Omega$, i.e. the explicit scope $\mathcal{I}_{\Omega} = \mathcal{I}$. However, if we assume that the type extension $\mathcal{I}(\mathcal{C})$ also contains type Integer, the $\mathcal{I}(\mathcal{C})$-object domain of $\mathcal{I}$ will also have integer values, and thus, $\mathcal{I}_{\Omega}(\mathcal{O})$ will not have any of them. For the same reason, there is a trivial function $K^{\mathcal{I}_{\Omega}}(\Omega)$ between the formula $\Omega$ and the interpretation $\mathcal{I}$. 

61
Consider the formula $\Omega$ defined before and the multiplicity constraints $\Phi$ in Section 3.1. We can now use the extended CSMOF institution to prove $\Omega \models \Sigma \Phi$. We need to find an interpretation $I$ such that $I \models \Omega \models \Sigma \Phi$. As shown in Section 3.1, the interpretation $I$ satisfies the multiplicity constraints $\Phi$, and also the SW-model $\Omega$, as defined before. But the definition of $\Omega \models \Sigma \Phi$ is not given for a single interpretation $I$, i.e. it holds if for all interpretations $I$, we have that $I \models \Omega \models \Sigma \Phi$. This holds only for multiplicity constraints $\Phi$ involving the same types defined in a SW-model $\Omega$ (those which are not affected by $|\Omega|$). For any other multiplicity constraint we can find an interpretation such that it satisfies $\Omega$ but not $\Phi$, which is correct.

Let’s consider an example in which there is a signature with types $A$, $B$ and $C$, and associations between them. There is also an interpretation $I$ with just two related elements, one of type $A$ and another of type $B$. We also have two formulas: a multiplicity constraint formula $\varphi = B \cdot re = 1$, i.e. there must be exactly one element of type $C$ related with any element of type $B$, and a SW-model formula $\Omega = A^A \oplus B^B \oplus \langle ra, a, rb, b \rangle$. We can notice that the interpretation provides a semantic representation for the SW-model, i.e. $I \models \Omega$ but $I \not\models \varphi$, since there is no semantic element of type $C$. This is the counterexample to prove that $\Omega \not\models \varphi$.

We can now consider a set of formulas $\Psi_1$ containing a formula $\Omega_1$ corresponding to the class SW-model in Figure 1c, and the set of multiplicity constraint formulas, as defined before. Moreover, we can consider a set of formulas $\Psi_2$ composed by a formula $\Omega_2$ corresponding to the relational SW-model in Figure 1d, and the set of multiplicity constraint formulas derived from the relation metamodel in Figure 1b, which were not defined in past sections. As defined in Section 4, we can also have a set of QVTR formulas $\varphi$ representing keys and transformation rules and an interpretation $M = (M_1^{\Omega_1}, M_2^{\Omega_2})$, which satisfies these formulas. With all this information we can prove that the QVTR formulas $\varphi$ are implied by the SW-models in $\Psi_1$ and $\Psi_2$, i.e. $\Psi_1 \cup \Psi_2 \models \Sigma \varphi$, since there exists the interpretation $M$ such that $M \models \Sigma \varphi_1$ and $M \models \Sigma \varphi_2$.

6. An Environment for Verification

Now that we have defined formal foundations for the MDE elements, we show how these foundations can be used for the definition of an environment for verification using heterogeneous verification approaches, following the ideas in [13, 12]. We already explained how a separation of duties in different technological spaces is carried out for dealing with the specification and formal verification in the context of MDE, as is graphically depicted in Figure 4. On one side there are those experts in the MDE domain specifying models and transformations, and on the other, those in formal methods (FM) conducting the verification process. Our proposal is to exploit the Theory of Institutions as a sound basis for constructing a FM technological space in which several logics can be used for verification [16]. Although this idea can be potentially formalized for any transformation approach and language, our proposal is aligned with the OMG standards, in particular the MOF and the QVT-Relations languages.

In the Sections 3.1 and 4 we provided the institutions CSMOF for SW-models, metamodels and the conformance relation between them, and QVTR for model transformation. In Section 5 we defined an extension of these institutions in order to be used within a proof environment. The satisfaction relations express very basic correctness criteria which must be ensured by a model transformation engine. As in [32], they are focused on checking partial correctness of declarative, rule-based transformations between constrained metamodels. These institutions (represented in the leftmost box of the FM technological space in Figure 4) provide a generic representation of the MDE elements with formal semantics.

In order to use them for verification purposes we need first to address the partial correctness checking, and also supplement the MDE elements with additional properties to be checked in several semantic domains. For the first requirement, we do not define a proof calculus for the institutions, but we follow the same solution as for the second requirement: we translate the MDE elements into other semantic domains (also defined as institutions) and re-use their entailment systems. This is called borrowing of an entailment system. Two institutions may be related through institution morphisms [33] which come in several flavors. In particular, there are so-called institution comorphisms which capture how a weaker and poorer institution can be represented in a stronger and richer one. In some cases, it is possible to borrow the entailment systems of an institution via a comorphism, i.e. to translate the proof goals of the source institution via the comorphism, and then use the sound proof calculus of the target institution. The definition of comorphisms not only gives us the chance of using some proof calculus, but also of supplementing the former specification of MDE elements with additional properties using the host logic. To the extent that there are many logics connected through comorphisms, the capabilities of the environment increase. As an example, we can
define comorphisms from our extended institutions to the Common Algebraic Specification Language (CASL, [34]), a general-purpose specification language (in the center box of the FM technological space in Figure 4). If we have a specification of the UML class diagrams metamodel in Figure 1a which is expressed with the institution CSMOF, we can translate the specification into CASL and supplement it with the OCL constraint expressed in first-order logic, addressing semantical conformance. The main property with respect comorphisms is that we ensure that the semantics is preserved through the translation.

These theoretical ideas must be addressed in practice with tool support. The importance of CASL relies in that it is the main language within the Heterogeneous Tool Set (HETS, [35, 12]), which is a tool that supports heterogeneous multi-logic specifications (represented in the rightmost box of the FM technological space in Figure 4). HETS allows defining institutions and comorphisms, and also provides proof management and monitoring capabilities for heterogeneous specification, when 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) which can be accessed through comorphisms. We can provide HETS with specific institutions for the specification of MDE elements. With this approach we have only one generic representation of the MDE elements which is formally (and automatically) translated into CASL or any other connected logic when needed, and those logics can also be used to specify additional verification properties which must be addressed, and perform the verification assisted by the tool. In most cases a general-purpose logic, as provided by CASL, is enough to cover most of the verification approaches in [36]. However, automatic proofs are not always possible, and in this sense within HETS it is possible to choose not to use an automated theorem proving system, but for example an interactive theorem prover.

6.1. Benefits and Limitations

Our proposal has many benefits from a software engineering perspective. The first quality attribute we can remark is usability. The MDE expert specifies a model transformation using QVT-Relations and such specification is taken by the formal verification expert who defines the formal properties to be verified. HETS provides a graphical user interface that can be used for visualizing the whole proof and selecting a prover for the corresponding logic. If the proof is truly heterogeneous (i.e. there is more than one logic involved in the specification) the tool performs automatic translations of proof obligations into other logics and allows selecting the corresponding prover to be used. In order to do this there must be a comorphism linking the current logic with the others involved. The definition of comorphisms may neither be direct nor even possible. However, the definition of a comorphism into CASL provides others already defined as a graph of logics within HETS. In this way CASL can be used as a pivot language for specification.

The environment supports a separation of duties between software developers such that a formal perspective is available whenever it is required. The verification process is performed hiding the theoretical details of the environment from the MDE experts. This means that from a FM perspective, the use of the environment is the same as the use
of any other semantic domain. In particular, the use of CASL is equivalent to the use of first-order logic. Moreover, we could perform an automated verification of some properties (e.g. those related to the basic correctness criteria defined before) by running HETS in the background and providing a better user interface to show the problems found. For this to be possible, we have to work on improving feedback from existing formal tools. It is also possible to add tools already used in related work for verification. None of those aspects are trivial so further study is needed.

The implementation of comorphisms allows us to move any specification from one logic to another, so there is no need of rewriting the MDE elements in each logic involved. This makes the environment scalable in terms of the transformation specification. However, an aspect which is not scalable is the size of the problem, since bigger specifications are, from a FM perspective, harder to verify, especially when proofs cannot be automated. When the representation of MDE elements is translated into a specific domain, it conforms a transformation model [37]. This SW-model states how elements are related, and these relations allow answering different syntactic and semantic questions.

The environment is also extensible since it potentially supports any kind of logic and allows the inclusion of new logics. This allows a wide variety of verification approaches so that FM experts can work in the domain in which they feel more confident, or choose the tool they prefer. The use of several domains is useful not only for verification but also for specification purposes, e.g. for the connection of MDE elements with more traditional software artifacts in a non-full MDE development. As stated before, the inclusion of a new logic or the definition of a comorphism requires the development of formal definitions which may neither be direct nor even possible.

Moreover, although our proposal is aligned with OMG standards, this idea can be potentially formalized for any transformation approach and language, which allows extending the approach as far as necessary. However, further studies are needed in this sense. Finally, the environment is reliable since it is supported by a well-founded theory and by a mature tool in which there are several logics already defined.

7. Related Work

There are many proposals 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. rewriting logic [26, 38], constructive type theory [39, 10], first-order logic [32, 40]. Unlike them, we prefer to define a generic institution not restricted by any logical domain. In [24] the authors use a graph-based representation for metamodels and SW-models which is commonly used in the field of graph transformations. In [41] the author proposes a formal approach for the definition of metamodels (not based on MOF) using a meta-notation called GEBNF (graphic extension of BNF) and the specification of constraints on SW-models in a formal logic language induced from it. This is also a shallow embedding, but in this case the author provides an institution in which signatures represent metamodels (in GEBNF), interpretations represent SW-models, and formulas are predicates using the language induced from GEBNF (intended to be as expressive as OCL). In [42] the author proposes an algebraic formalization of MOF metamodels based on the NEREUS language, which can be viewed as an intermediate notation that can be translated to other formal languages. Something similar is presented in [43] in which the authors define a method for representing MOF metamodels in a formalism called Alloy, based on first-order logic. In [21, 22] the authors define institutions for simple and stereotyped UML Class Diagrams. We have adapted these proposals for the definition of our CSMOF institution.

With respect to model transformations, the semantics of QVT-Relations was represented in terms of a shallow embedding, e.g. into rewriting logic [8] and coloured petri nets [44]. There are also embeddings into specific tools, as in the case of Alloy [45] and KIV [46], which provide model checking capabilities. As said before, we do not follow this approach since a unified mathematical formalism can be quite restrictive. In [47] the authors define an institution for graph transformation systems, not completely related to QVT-Relations or any other transformation language. They interpret metamodels and SW-models as graphs, as we presented in Section 3.2. In [31] the authors present a formal semantics for the QVT-Relations check-only scenario based on algebraic specification. The definition of the institution is much more complex than ours, and the work does not envisons a scenario in which the elements of the transformation are translated to other logics for verification. In [48] the authors define game-theoretic semantics of QVT-Relations check-only transformations. This semantics is devised for analyzing the implications of minor variations in decisions about what the meaning of a QVT-R transformation should be.

There are some proposals for the comprehensive verification of MDE elements based on a unified mathematical formalism, e.g. [38, 8] in which rewriting logic is used to analyze MOF-like and QVT-like elements. Since our
institutions are logic-independent, they provide more flexibility for the definition of further specific translations into other logics and languages. As far as we know, there is no other comprehensive institutional approach. However, algebraic specifications and institutions were used in similar contexts. In [49] the authors propose to define concrete institutions for any specific metamodel involved in a transformation, which is extremely expensive from a practical perspective. They also state the correctness of a transformation in terms of the existence of an intermediate institution that relates the source and target institutions through some institution morphism. This is somehow restrictive since it assumes a semantic relation between metamodels, and not every model transformation is semantic-preserving. In [50] the authors provide an algebraic representation of MDE elements in which metamodels are represented as many-sorted algebras. They broadly define how these settings are related to institutions but they do not define a specific institution for metamodels. The authors also define model-to-model transformations based on triple algebras (in consonance with relational model transformations). The representation differs from ours since it depends on the formal definition of a triple algebra. In [13] the authors define a heterogeneous approach to the semantics of UML. This work defines the institution for UML class diagrams in which our CSMOF institution is based.

It is also possible to define only one institution for representing MDE elements. The CSMOF institution together with an extended definition of formulas adding an expressions language on metamodels can be enough. Indeed, this representation is equivalent to the idea of a transformation contract, as in [51], in which both source and target metamodels are defined in a unified SW-model, and constraints are used for expressing invariants in metamodels as well as describing the pre and post-conditions of the transformation. This is a complementary approach that deserves further study. Other proposals are based on the construction of a unified representation of the MDE elements using OCL contracts [37]. In [52, 53] the authors present how to extract OCL contracts from several model transformation languages. These contracts can be easily checked and analyzed using available OCL model finders. Moreover, in [9] the authors define a language-independent representation of metamodels and model transformations supporting many transformation languages. They also define mappings to the B and Z3 formalisms. Since they use only one generic language, only one semantic mapping needs to be defined for each target formalism. However, the semantic mapping should be semantics-preserving, and this aspect is not formally addressed in such work. In our case, the use of institutions allow the definition of comorphisms which by definition preserve the semantics with respect to the satisfaction relation. Moreover, by defining a comorphism into CASL and the corresponding implementation in HETS, it is possible to connect our institutions to several logics and tools.

8. Conclusions and Future Work

In this paper we have defined a formalization of the MDE elements based on the Theory of Institutions. We have provided institutions for representing the structural conformance relation between SW-models and metamodels specified with a simplified version of MOF (which we called CSMOF), and for QVT-Relations model transformations (which we called QvTR). These institutions provide a generic representation of the MDE elements with formal semantics covering almost every important element. We also studied how to represent semantical conformance with respect to OCL constraints. However, the fine tuning of an institution for OCL is part of our future work. The formalization improves existent knowledge on the semantics of MDE and on the use of institutions for the formalization of specification languages. Indeed, those representations could be connected with other UML languages to conform the heterogeneous institution environment in the sense of [13].

We have also shown how the formalization supports the definition of a comprehensive environment for the formal verification of MDE elements using heterogeneous verification approaches. In this environment the MDE elements are defined without depending on any specific logical domain, and semantic-preserving translations can be defined from these elements to several other logical domains where the verification of desired properties should be addressed. This is possible by using the Theory of Institutions. The idea is innovative since in most of the cases the MDE elements are directly represented within some specific logical domain which makes their translation into another domain expensive. In our case we do not need to maintain multiple formal representations of the same MDE elements, but to define those translations which can be then reused. These translations are also useful to integrate MDE elements with the specification and verification of other software artifacts in a traditional software development.

Although we build on the MOF and QVT-Relations standards for the specification of the MDE elements, we follow an idea general enough to be extended to other transformation approaches and languages, which is devised as future work. Another alternative is the definition of an institution for a generic transformation language, as in [9], or
a transformation contract, as in [51]. In both cases the institution must express the intended effect of a transformation by a metamodel plus constraints (e.g. in OCL) describing the pre and post-conditions of the transformation. We must conduct a deeper study in this sense.

Our medium-term goals are the implementation of the environment within HETS by providing comorphisms from our institutions to CASL. With this, MDE experts can specify model transformations in their technological space and such specifications can be complemented by verification experts with other properties to be verified, e.g. non-structural constraints. All this information is taken by HETS, which performs automatic translations of proof obligations into other logics and allows selecting the corresponding prover to be used, whilst a graphical user interface is provided for visualizing the whole proof. In other words, we provided to MDE practitioners the “glue” they need for connecting their technological space with the logical domains needed for verification. The existent connections between CASL and other logics, broadens the spectrum of logical domains in which the verification can be addressed.

Acknowledgements

We want to thank María Victoria Cengarle and Alexander Knapp for their comments on the contents of this paper.

References


