Model-Driven Engineering in the Heterogeneous Tool Set

Daniel Calegari[1](✉), Till Mossakowski[2], and Nora Szasz[3]

[1] Universidad de la República, Montevideo, Uruguay
dcalegar@fing.edu.uy
[2] Otto-von-Guericke University Magdeburg, Magdeburg, Germany
mossakow@iws.cs.uni-magdeburg.de
[3] Facultad de Ingeniería, Universidad ORT Uruguay, Montevideo, Uruguay
szasz@ort.edu.uy

Abstract. We have defined a unified environment that allows formal verification within the Model-Driven Engineering (MDE) paradigm using heterogeneous verification approaches. The environment is based on the Theory of Institutions, which provides a sound basis for representing MDE elements and a way for specifying translations from these elements to other logical domains used for verification, such that formal experts can choose the domain in which they are more skilled to address a formal proof. In this paper we present how this environment can be supported in practice by the Heterogeneous Tool Set (Hets). We define semantic-preserving translations from the MDE elements to the core language of Hets, and we also show how it is possible to move from it to other logics, both to supplement the original specification with other verification properties and to perform a heterogeneous verification.

Keywords: Verification · Formal methods · Model-Driven Engineering

1 Introduction

The Model-Driven Engineering (MDE,[1]) paradigm is based on the construction of models representing different views of the system to be constructed, and model transformations as the main activity within the software development process. In this context, there are multiple properties that can be verified [2], from syntactic to semantic ones, and at different abstraction levels. Whenever formal verification is mandatory, there is a plethora of verification approaches with different objectives, formalisms and supporting tools, which are heterogeneous and not integrated. With an heterogeneous approach [3], different formalisms are used for expressing parts of a problem and semantic-preserving mappings allow the communication between these formalisms in order to compose different views to an overall specification of the whole problem. We have followed this approach by proposing a theoretical environment for the formal verification of different MDE aspects using heterogeneous verification approaches [4], based on the theory of

C. Braga and N. Martí-Oliet (Eds.): SBMF 2014, LNCS 8941, pp. 64–79, 2015.
DOI: 10.1007/978-3-319-15075-8_5

Institutions [5]. This environment proposes a generic representation of the MDE elements (by means of institutions) which can be formally (and automatically) translated into other formalisms, providing the "glue" that formal experts need to choose the formalism in which they are more skilled to address a formal proof.

In this paper we show how the environment can be supported in practice using the Heterogenous Tool Set (Hets,[3]) , which is meant to support heterogeneous multi-logic specifications. It also provides proof management capabilities for monitoring the overall correctness of a heterogeneous specification whereas different parts of it are verified using (possibly) different formalisms. We first define from a theoretical perspective how MDE elements can be integrated in this tool by defining semantic-preserving translations to the Common Algebraic Specification Language (Casl,[6]), which is the core language of Hets. The existent connections between Casl and other formalisms broadens the spectrum of formal domains in which verification can be addressed. We also detail the implementation of a prototype which allows us to specify MDE elements, supplement them with multi-logic properties, and perform a heterogeneous verification.

The remainder of the paper is structured as follows. In Section 2 we introduce the main concepts of MDE based on a running example, and in Section 3 we summarize how these elements can be represented within our institution-based environment. Then, in Section 4 we present how this environment can be formally connected with Casl, and in Section 5 we give details about an implementation of these ideas using Hets. Finally, in Section 6 we discuss related work and in Section 7 we present some conclusions and an outline of further work.

2 Model-Driven Engineering

In MDE there are two key elements: models specifying different views of the system to be constructed and model transformations allowing the (semi)automatic construction of the system by processing the models.

Every model *conforms* to a metamodel which introduces the syntax and semantics of certain kinds of models. The MetaObject Facility (MOF, [7]) is a standard language for metamodeling, basically defining hierarchical-structured classes with properties that can be attributes (named elements with an associated primitive type or class) or associations (relations between classes in which each class plays a role within the relation). Every property has a multiplicity which constraints the number of elements that can be related through it. If there are conditions that cannot be captured by the structural rules of this language, the Object Constraint Language (OCL, [8]) is used to specify them. These considerations allow defining conformance in terms of *structural* and *non-structural* conformance. Structural conformance with respect to a metamodel means that in a given model every object and link is well-typed and the model also respects the multiplicity constraints. Non-structural conformance means that a given model respects the invariants specified with the supplementary language.

Consider as an example a simplified version of the well-known Class to Relational model transformation [9]. The metamodel in the left side of Fig. 1 defines

UML class diagrams, where classifiers (classes and primitive types) are contained in packages. Classes can contain one or more attributes and may be declared as persistent, and each attribute is typed by a primitive type. Notice that a class must contain only one or two attributes, and also that the Classifier class is not abstract. We handle these aspects differently from UML class diagrams in order to have a more complete example. In the right side of Fig. 1 there is a model composed by a persistent class of name `ID` within a package of name `Package`. The class has an attribute of name `value` and type `String`.

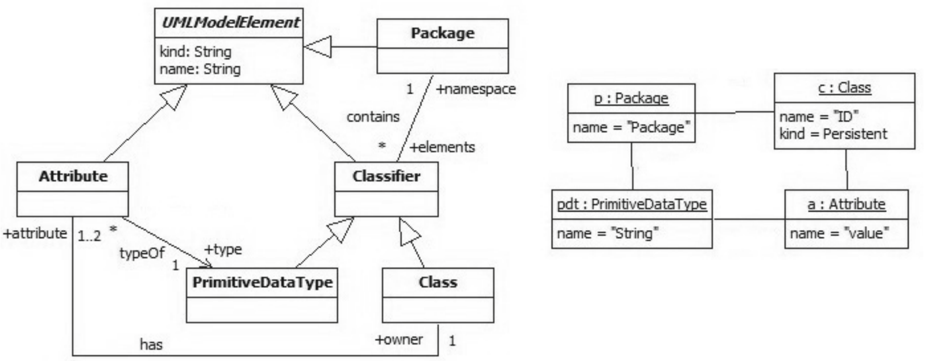

Fig. 1. Class metamodel and model of the example

A model transformation takes as input a model conforming to certain metamodel and produces as output another model conforming to another metamodel (possibly the same). Query/View/Transformation Relations (QVT-Relations, [9]) is a relational language which defines transformation rules as mathematical relations between source and target elements. A transformation is a set of interconnected relations: top-level relations that must hold in any transformation execution, and non-top-level relations that are required to hold only when they are referred from another relation. Every relation defines a set of variables, and source and target patterns which are used to find matching sub-graphs of elements in a model. Relations can also contain a `when` clause which specifies the conditions under which the relationship needs to hold, and a `where` clause which specifies the condition that must be satisfied by all model elements participating in the relation. The `when` and `where` clauses, as well as the patterns may contain arbitrary boolean OCL expressions and can invoke other relations.

The transformation of the example basically describes how persistent classes within a package are transformed into tables within a schema, and attributes of a class are transformed into columns of the corresponding table. Below we show an excerpt of this transformation. There are keys defined as the combination of those properties of a class that together can uniquely identify an instance of that class, e.g. there are no two tables with the same name within the same schema.

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

  top relation PackageToSchema { ... }

  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
    };
    when { PackageToSchema(p, s); }
    where { AttributeToColumn(c, t);}
  }

  relation AttributeToColumn { ... }
}
```

3 An Institution-Based Environment for MDE

Our environment [4] is based on representing models (from now on SW-models to avoid confusion), metamodels, the conformance relation, transformations and verification properties in some consistent and interdependent way without depending on any specific logical domain. We follow an heterogeneous specification approach [3] which is based on providing *Institutions* [5] for representing the syntax and semantics of the elements. An institution is defined as:

- a category Sign of signatures (vocabularies for constructing sentences in a logical system) and signature morphisms (translations between vocabularies)
- a functor Sen : Sign → Set giving a set of sentences for each signature and a function $\mathrm{Sen}(\sigma)$:$\mathrm{Sen}(\Sigma_1) \to \mathrm{Sen}(\Sigma_2)$ translating formulas to formulas for each signature morphism $\sigma : \Sigma_1 \to \Sigma_2$;
- a functor $\mathrm{Mod} : \mathrm{Sign}^{op} \to \mathbf{Cat}$, giving a category $\mathrm{Mod}(\Sigma)$ of models (providing semantics) for each signature Σ and a reduct functor $\mathrm{Mod}(\sigma)$:$\mathrm{Mod}(\Sigma_2) \to \mathrm{Mod}(\Sigma_1)$ translating models to models (and morphisms to morphisms) for each signature morphism;
- a satisfaction relation of sentences by models, such that when signatures are changed (by a signature morphism), satisfaction of sentences by models changes consistently, i.e. $M_2 \models_{\Sigma_2} \mathrm{Sen}(\sigma)(\varphi)$ iff $\mathrm{Mod}(\sigma)(M_2) \models_{\Sigma_1} \varphi$

We provide an institution $\mathcal{I}^{\mathrm{Q}}$ for QVT-Relations check-only unidirectional transformations (which we called Qvtr). This institution needs a representation of SW-models and metamodels, therefore we define an institution $\mathcal{I}^{\mathrm{M}}$ for the structural conformance relation between them based on a simplified version of MOF (which we called Csmof). Complete definitions can be found in [10].

The institution $\mathcal{I}^{\mathrm{M}}$ represents the MOF-based structural conformance relation between metamodels and SW-models. From any metamodel we can derive a signature $\Sigma = (\boldsymbol{C}, \alpha, \boldsymbol{P})$ declaring: a finite class hierarchy $\boldsymbol{C} = (C, \leq_C)$ (a partial order between classes representing the inheritance relation between them) extended with a subset $\alpha \ \subseteq C$ denoting abstract classes; and a properties declaration (attributes and associations) $\boldsymbol{P} = (R, P)$ 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 $\langle r_1 : c_1, r_2 : c_2 \rangle$ representing a property and its opposite. The type c_i attached to the role r_i represents the type of the property, as well the type in the opposite side represents its owned class. By $\boldsymbol{T}(\boldsymbol{C})$ we denote the *type extension* of $\boldsymbol{C}$ by primitive types (e.g. Boolean) and type constructors (e.g. List). Formulas represent multiplicity constraints determining whether the number of elements in a property end is bounded (upper and/or lower). They are defined as follows: $\Phi ::= \#C \bullet R = \mathbb{N} \ \mid \ \mathbb{N} \leq \#C \bullet R \mid \ \#C \bullet R \leq \mathbb{N}$ The #-expressions return the number of links in a property when some role is fixed. The $\bullet$ operator represents the selection of the elements linked with another of class C through a role in R. An interpretation $\mathcal{I}$ (or model) contains a semantic representation for a SW-model, i.e. objects and links. It consists of a tuple $(\boldsymbol{V}^{\boldsymbol{T}}_{\boldsymbol{C}}(\boldsymbol{O}), \boldsymbol{A})$ where $\boldsymbol{V}^{\boldsymbol{T}}_{\boldsymbol{C}}(\boldsymbol{O}) = (V_c)_{c \in T(C)}$ is a $\boldsymbol{T}(\boldsymbol{C})$-object domain (a family of sets of object identifiers), $\boldsymbol{A}$ contains a relation $\langle r_1 : c_1, r_2 : c_2 \rangle^{\mathcal{I}} \subseteq V_{c_1} \times V_{c_2}$ for each relation name $\langle r_1 : c_1, r_2 : c_2 \rangle \in P$ with $c_1, c_2 \in T(C)$, and $c_2 \in \alpha$ implies $O_{c_2} = \bigcup_{c_1 \leq_C c_2} O_{c_1}$. Finally, an interpretation $\mathcal{I}$ satisfies a formula φ with some $c \bullet r$ if for any object of class c, the number of elements within $\mathcal{I}$ related through the role r (of a property of the class c) satisfies the multiplicity constraints. The satisfaction relation checks the multiplicity requirements of the structural conformance relation.

The institution $\mathcal{I}^{\mathrm{Q}}$ represents QVT-Relations transformations by extending the CSMOF institution. A signature is a pair $\langle \Sigma_1^{\mathrm{M}}, \Sigma_2^{\mathrm{M}} \rangle$ representing the source and target metamodels of the transformation, and an interpretation is a tuple $\langle \mathcal{M}_1^{\mathrm{M}}, \mathcal{M}_2^{\mathrm{M}} \rangle$ of disjoint $\mathrm{Sign}_i^{\mathrm{M}}$-interpretations that contains a semantic representation for the source and target SW-models. A formula φ^{K} represents a key constraint of the form $\langle c, \{r_1, ..., r_n\} \rangle$ $(1 \leq n)$ with $c \in C_i$ (i = 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). Roles determine the elements within these properties that together can uniquely identify an instance of the class. A formula φ^{R} represents a set of interrelated transformation rules, such that, given variables $X^s = (X^s)_{s \ \in (\bigcup_i T(C_i))}$, the formula is a finite set of tuples representing rules of the form $\langle \mathrm{top}, \mathrm{VarSet}, \mathrm{ParSet}, \mathrm{Pattern}_1, \mathrm{Pattern}_2, \mathrm{when}, \mathrm{where} \rangle$, where top $\in \{true, false\}$ defines if the rule is a top-level relation or not, VarSet $\subseteq X^s$ is the set of variables used within the rule, ParSet $\subseteq$ VarSet representing the set of variables taken as parameters when the rule is called from another one, $\mathrm{Pattern}_i$ $(i = 1, 2)$ are the source and target patterns, and when/where are the `when`/`where` clauses of the rule, respectively. A pattern is a tuple $\langle E_i, A_i, Pr_i \rangle$ such that $E_i \subseteq (X^c)_{c \ \in C_i}$ is a set of class-indexed variables, A_i is a set of elements representing associations of the form $rel(p, x, y)$ with $p \in P_i$ and $x, y \in E_i$, and Pr_i is a predicate over these elements. A `when`/`where` clause is a pair $\langle \text{when}_c, \text{when}_r \rangle$ such that when_c is a predicate with variables in VarSet, and when_r is a set of pairs of transformation rules and their parameters. The satisfaction relation expresses that the target SW-model is the result of transforming the source SW-model (both within the interpretation) according to the transformation rules and also that key constraints hold (both represented as formulas).

Institutions can be formally connected by means of (co)morphisms. Then, by defining these semantic-preserving translations, it is possible to connect MDE elements to potentially several logics for formal verification. In this way, we just specify MDE elements once, then spread this information into other logics to supplement this specification with additional properties, and finally choose the verification approach we want to use. To the extent that there are many logics connected through comorphisms, the capabilities of our environment increases. The environment supports a separation of duties between software developers (MDE and formal methods experts) such that a formal perspective is available whenever it is required. Moreover, comorphisms can be automated, as we show in the following sections, thus the environment is scalable in terms of the rewriting of MDE elements in each logic. Although our proposal is aligned with OMG standards, this idea can be potentially formalized for any transformation approach and language. This allows extending the approach as far as necessary.

4 Borrowing Proof Capabilities

We make use of the possibility of connecting our institutions to potentially several host logics, each one with its own proof system. The host logic allows both to supplement the information contained within the MDE elements with properties specified in the host logic, and to borrow its proof calculus for formal verification. For this, we us *generalized theoroidal comorphisms* (GTC,[11]). A GTC between two institutions $\mathcal{I}$ and $\mathcal{J}$consists of a functor $\Phi : \text{Th}^{\mathcal{I}} \rightarrow \text{Th}^{\mathcal{J}}$ translating theories (pairs of signatures and set of sentences), and a natural transformation $\beta : (\Phi)^{op}; \text{Mod}^{\mathcal{J}} \rightarrow \text{Mod}^{\mathcal{I}}$ translating models in the opposite direction.

We do not define GTC from the institutions defined in the last section, but from extended institutions $\mathcal{I}^{\text{M}^+}$ and $\mathcal{I}^{\text{Q}^+}$. We extend the definition of CSMOF formulas with a syntactic representation of SW-models as follows:

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

with $x^c \in X^c$ a variable representing a typed element, $\langle r_1, {x_1}^{c_1}, r_2, {x_2}^{c_2} \rangle$ representing a link between two typed elements with their respective roles, and $\Omega \oplus \Omega$ the composition of these elements. In the case of QVTR, we extend QVTR formulas by including extended CSMOF formulas, i.e. now there is a representation of multiplicity constraints and SW-models, indexed by the institutions in which they are defined. These extensions make it possible to use a proof system such that it is possible to prove that constraints (as a formula) are derived from a syntactic representation of a SW-model, which is the context where the verification must be done. An exhaustive discussion on this topic can be found in [10].

We defined GTCs from our extended institutions to Casl, a general-purpose specification language. The institution $\mathcal{I}^{\mathrm{C}}$ underlying Casl is the sub-sorted partial first-order logic with equality and constraints on sets $SubPCFOL^{=}$, a combination of first-order logic and induction with subsorts and partial functions. Since Casl has a sound proof calculus, and our comorphisms admit borrowing of entailment [3], we can translate our proof goals using the comorphism into Casl and use its proof calculus also for proving properties of our extended Csmof and Qvtr specifications. The importance of Casl is that it is the main language within the Heterogenous Tool Set (Hets, [3]), a tool meant to support heterogeneous multi-logic specifications. This comorphism not only allows us to have tool support for the verification of model transformation by using Hets (as will be introduced in Section 5) but also to move between the graph of logics within Hets to take advantage of the benefits of each logic.

In what follows we introduce Casl and resume the encoding of the main components of the extended institutions into it. An example of the encoding is given in Section 5, and a complete version can be found in [10].

4.1 Common Algebraic Specification Language

The institution $\mathcal{I}^{\mathrm{C}}$ for Casl is defined as follows. Signatures consist of a set S of sorts with a subsort relation $\leq$ between them, together with a family $\{PF_{w,s}\}_{w \in S^*, s \in S}$ of partial functions, $\{TF_{w,s}\}_{w \in S^*, s \in S}$ of total functions and $\{P_w\}_{w \in S^*}$ of predicate symbols. Signature morphisms consist of maps taking sort, function and predicate symbols respectively to a symbol of the same kind in the target signature, and they must preserve subsorting, typing of function and predicate symbols and totality of function symbols.

For a signature Σ, terms are formed starting with variables from a sorted set X using applications of function symbols to terms of appropriate sorts, while sentences are partial first-order formulas extended with *sort generation constraints* which are triples (S', F', σ') such that $\sigma' : \Sigma' \to \Sigma$ and S' and F' are respectively sort and function symbols of Σ'. Models interpret sorts as non-empty sets such that subsorts are injected into supersorts, partial/total function symbols as partial/total functions and predicate symbols as relations.

The satisfaction relation is the expected one for partial first-order sentences. A sort generation constraint (S', F', σ') holds in a model M if the carriers of the reduct of M along σ' of the sorts in S' are generated by function symbols in F'.

4.2 Encoding Csmof into Casl

We define a GTC between the extended Csmof institution $\mathcal{I}^{\mathrm{M}^+}$ and the institution $\mathcal{I}^{\mathrm{C}}$ for $SubPCFOL^{=}$. The class hierarchy represented within a $\mathcal{I}^{\mathrm{M}^+}$ signature is basically translated into a set of sorts complying with a subsorting relation, properties are translated into predicates, and an axiom is introduced to relate predicates derived from bidirectional properties. Formally, every $\mathcal{I}^{\mathrm{M}^+}$ signature $\Sigma = (\boldsymbol{C}, \alpha, \boldsymbol{P})$ with $\boldsymbol{C} = (C, \leq_C)$ and $\boldsymbol{P} = (R, P)$ is translated into a theory $((S, TF, PF, P, \leq_S), E)$ such that:

- For every class name c in C, there is a sort name $c \in S$.
- For every $c_1 \leq_C c_2$ with $c_1, c_2 \in C$, we have $c_1 \leq_S c_2$ with $c_1, c_2 \in S$.
- For every $c \in \alpha$ there is an axiom in E stating that c is the disjoint embedding of its subsorts (sort generation constraint).
- For every $\langle r_1 : c_1, r_2 : c_2 \rangle \in P$, there are two predicates $r_1 : c_2 \times c_1$ and $r_2 : c_1 \times c_2 \in \Pi$, and an axiom in E stating the equivalence of the predicates, i.e. $r_1(x, y)$ iff $r_2(y, x)$ with $x \in S_1, y \in S_2$. In the case of predicates with the default role name _, we only generate the predicate in the opposite direction of the default role, i.e. if $\langle \_ : c_1, r_2 : c_2 \rangle$ or $\langle r_1 : c_1, \_ : c_2 \rangle$ we only have $r_2 : c_1 \times c_2$ or $r_1 : c_2 \times c_1$, respectively.

We consider the existence of a built-in extension of the institution $\mathcal{I}^{\mathrm{C}}$, e.g. the CASL standard library. The sets of functions TF and PF within this extension contain those functions defined for built-in types (like + for strings).

As an example, the signature corresponding to the class metamodel in Fig. 1 is translated into a theory such that there are sorts for each class, e.g. Package and UMLModelElement, within the subsorting relation, e.g. Package $\leq_S$ UMLModelElement; and there are predicates for each property, e.g. elements : Package × Classifier and name : UMLModelElement × String. There is a sort generation constraint stating that UMLModelElement is the disjoint embedding of its subsorts Attribute, Classifier, and Package. There are also axioms stating the equivalence of the predicates derived from bidirectional properties, e.g. $\forall\, x :$ Package, $y :$ Classifier. elements$(x, y) \Leftrightarrow$ namespace(y, x)

In the case of a SW-model formula Ω, each variable within the formula (representing an object) is translated into a total function of the corresponding type. We also add several axioms in order to represent implicit constraints in the $\mathcal{I}^{\mathrm{M}^+}$ institution which are not necessarily kept when representing the basic elements in $SubPCFOL^=$, as for example the need of distinguishing between two different variables (functions in the target institution) and the specification of the cases in which a property holds (when there is a syntactic link represented within the formula Ω). Formally,

- For every $x^c \in \upsilon(\Omega)$ there is a total function $x : c \in TF$ with $c \in S$
- For every $\langle r_1, x^{c_1}, r_2, y^{c_2} \rangle \in \omega(\Omega)$ with $\langle r_1 : c_1, r_2 : c_2 \rangle \in P$, there is an axiom in E stating that the predicate $r_2 : c_1 \times c_2$ holds for $x : c_1, y : c_2 \in TF$. Notice that the opposite direction holds by the equivalence of predicates stated during the signature translation.
- E has some additional axioms:
  - Distinguishability: $\{x_i \neq x_j \mid i \neq j.\ x_i, x_j : c \in TF\}$ for all $c \in S$
  - Completeness of elements: for all $x : c$ we have that $x = o_i$ for some $o_i : c \in TF$. When c is a non-abstract class having sub-classes, completeness must be defined for $o_i : c' \in TF$ for all $c' \leq c$.
  - Completeness of relations: for all $x : c_1, y : c_2$ we have that $r(x, y)$ holds only if $x = o_1$ and $y = o_2$ for some $o_1 : c_1, o_2 : c_2$ for which $r(c_1, c_2)$ hold.

The "distinguishability" and "completeness of elements" axioms correspond to the "no junk, no confusion" principle: there are no other values than those denoted by the functions $x : c$, and distinct functions denote different values.

The variables within the class SW-model in Fig. 1 are translated into total functions, e.g. p : Package, c : Class and ID : String. Moreover, for every link there is an axiom stating that the corresponding predicate holds for the functions corresponding to the translated elements within the link. This axiom can be stated in conjunction with the "completeness of relations", e.g. $\forall\ x : \text{Package}, y : \text{Classifier}.\ \text{elements}(x, y) \Leftrightarrow (x = \text{p}\ \wedge\ y = \text{c}) \vee (x = \text{p}\ \wedge\ y = \text{pdt})$. In the case of the non-abstract class Classifier which has sub-classes, the "completeness of elements" constraint is stated by the axiom: $\forall\ x : \text{Classifier}.\ x = \text{c} \vee x = \text{pdt}$. Finally, the "distinguishability" constraint must be stated between elements of sorts related by the subsorting relation. For example, in the case of the elements within the UMLModelElement hierarchy, we have the following constraint: $\neg(\text{a} = \text{c}) \wedge \neg(\text{a} = \text{p}) \wedge \neg(\text{a} = \text{pdt}) \wedge \neg(\text{c} = \text{p}) \wedge \neg(\text{c} = \text{pdt}) \wedge \neg(\text{p} = \text{pdt})$.

For the translation of a multiplicity constraint formula we define the following predicates for constraining the size of the set of elements in a relation:

- $min(n, R : D \times C)$ holds if for all $y : D$ there exists $x_1, ..., x_n : C$ such that $R(y, x_i)$ for all $i = \{1..n\}$, and $x_i \neq x_j$ for all $i = \{1..n-1\}$, $j = i + 1$.
- $max(n, R : D \times C)$ holds if for all $y : D$ and $x_1, ..., x_{n+1} : C$, $Rel(y, x_i)$ for all $i = \{1..n + 1\}$ implies there is some $x_i = x_j$ with $i = \{1..n\}$, $j = i + 1$.

The first predicate states that there are at least n different elements related to every element y by the relation R, which represents a minimal cardinality for the relation. The other predicate states that there are no more than n elements related to any element y by the relation R, which represents a maximal cardinality for the relation. Using these predicates, we can translate any multiplicity constraint formula as follows:

- $n \leq \#D \bullet R$ is translated into $min(n, R : D \times C)$
- $\#D \bullet R \leq n$ is translated into $max(n, R : D \times C)$
- $\#D \bullet R = n$ is translated into $min(n, R : D \times C) \wedge max(n, R : D \times C)$

such that $Q : C \times D, R : D \times C \in \Pi$ are the predicates generated by the translation of the property $\langle R : C, Q : D\rangle$. If the multiplicity constraint involves the other end, i.e. $C \bullet Q$, the predicate $Q : C \times D$ is used instead of $R : D \times C$.

As an example, the formula $\#(\text{UMLModelElement}\ \bullet\ \text{name}) = 1$ derived from Fig. 1 is translated into the conjunction of

$$\begin{aligned} &min(1, \text{name} : \text{UMLModelElement} \times \text{String}) = \\ &\qquad \forall\ x_1 : \text{UMLModelElement}.\ \exists\ y_1 : \text{String}.\ \text{name}(x_1, y_1) \\ &max(1, \text{name} : \text{UMLModelElement} \times \text{String}) = \\ &\qquad \forall\ x_1 : \text{UMLModelElement}, y_2, y_1 : \text{String}. \\ &\qquad\quad (\text{name}(x_1, y_1) \wedge\ \text{name}(x_1, y_2)) \Rightarrow y_1 = y_2 \end{aligned}$$

Given a $\mathcal{I}^{\text{M}^+}$ theory $T = \langle \Sigma, \Psi \rangle$, a $\mathcal{I}^{\text{C}}$ model M of its translated theory (Σ', E) is translated into a Σ-interpretation denoted $I = (\boldsymbol{V_C^T}(\boldsymbol{O}), \boldsymbol{A})$ such that: each non-empty carrier set $|M|_s$ with $s \in S$, is translated into the set V_c in the object domain $\boldsymbol{V_C^T}(\boldsymbol{O})$, with s the translation of type $c \in T(C)$; and each relation p_M of a predicate symbol $r_2(c_1, c_2) \in P$ derived from the translation of a predicate $\langle r_1 : c_1, r_2 : c_2 \rangle$, is translated into the relation $p^{\mathcal{I}} \subseteq V_{c_1} \times V_{c_2}\ \in \boldsymbol{A}$.

4.3 Encoding QVTR into CASL

We define a GTC between the extended QVTR institution $\mathcal{I}^{Q^+}$ and the institution $\mathcal{I}^{C}$ for $\mathit{SubPCFOL}^{=}$. Every $\mathcal{I}^{Q^+}$ signature $\langle \Sigma_1^{M}, \Sigma_2^{M} \rangle$ is translated by the functor Φ into a theory such that each signature Σ_i^{M} is translated as defined in the encoding of CSMOF into CASL. We assume that the institution $\mathcal{I}^{E}$ of the expressions language has a correspondence (via a comorphism) with the built-in extension of the institution $\mathcal{I}^{C}$.

Formulas representing keys and transformation rules are translated into named first-order formulas. Formulas will be of the form $P \Leftrightarrow F$ such that P is the predicate naming the formula, and F represents the conditions which must hold in order to satisfy a key constraint φ^{K} or transformation φ^{R}.

In the case of a formula φ^{K}, the formula F defines that there are not two different instances of that class with the same combination of properties conforming the key of such class. Formally, any formula $\langle C, \{r_1, ..., r_n\} \rangle$ is translated into a predicate $\mathit{key\_C}$ naming a key constraint definition, and a formula of the form $\mathit{key\_C} \Leftrightarrow \forall x, y \ \in C, v_j : T_j.\ x \neq y \rightarrow \bigwedge_{i,j} r_i(x, v_j) \ \rightarrow \bigvee_{i,j} \neg r_i(y, v_j)$, with $r_i(\_, \_)$ one of the two predicates from the translation of the property $\langle r_1 : C_1, r_2 : C_2 \rangle$ such that one of the roles is of type C and the other of type T_j.

The key formula in the example is translated into the expression

$$\begin{aligned} &\mathrm{key\_Table} \Leftrightarrow \forall x, y \ \in \mathrm{Table}, v_1 : \mathrm{String},\ v_2 : \mathrm{Schema}. \\ &\quad x \neq y \rightarrow \mathrm{name}(x, v_1) \wedge \mathrm{schema}(x, v_2) \rightarrow \neg\mathrm{name}(y, v_1) \vee \neg\mathrm{schema}(y, v_2) \end{aligned}$$

In the case of a formula φ^{R}, the formula F declares that top-level relations must hold, and each individual rule is translated into the set of conditions stated by the checking semantics of QVT-Relations. Formally, every rule $\mathrm{Rule} = \langle \mathrm{top}, \mathrm{VarSet}, \mathrm{ParSet}, \mathrm{Pattern}_i \ (i = 1, 2), \mathrm{when}, \mathrm{where} \rangle \in \varphi^{R}$ is translated into: a predicate $\mathit{Rule} : T_1 \times ... \times T_n \in P$ with $\mathrm{ParSet} = \{T_1, .., T_n\}$, and a predicate $\mathit{Top\_Rule}$ without parameters (only if $\mathrm{top} = \mathit{true}$), naming the formula; and a formula $\forall v_1 : T_1, ..., v_n : T_n.\ \mathit{Rule}(v_1, ..., v_n) \Leftrightarrow F$ such that $\mathit{Rule}(v_1, ..., v_n)$ is the predicate defined before. In the case of a top rule, there is also a formula $\mathit{Rule} \Leftrightarrow F$. For the formula F there are two cases corresponding to the checking semantics of QVT-Relations:

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

$$\begin{aligned} &\forall\ x_1, ..., x_n \in \ (\mathrm{VarSet} \backslash 2\_\mathrm{VarSet}) \backslash \mathrm{ParSet}.\ (\Phi(\mathrm{Pattern}_1) \rightarrow \\ &\quad \exists\ y_1, ..., y_m \in 2\_\mathrm{VarSet} \backslash \mathrm{ParSet}.\ (\Phi(\mathrm{Pattern}_2)\ \wedge\ \Phi(\mathrm{where}))) \end{aligned}$$

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

$$\begin{aligned} &\forall\ z_1, ..., z_o \in \ \mathrm{WhenVarSet} \backslash \mathrm{ParSet}.\ (\Phi(\mathrm{when}) \rightarrow \\ &\quad \forall\ x_1, ..., x_n \in \ (\mathrm{VarSet} \backslash (\mathrm{WhenVarSet} \cup 2\_\mathrm{VarSet})) \backslash \mathrm{ParSet}. \\ &\qquad (\Phi(\mathrm{Pattern}_1) \rightarrow \exists\ y_1, ..., y_m \in 2\_\mathrm{VarSet} \backslash \mathrm{ParSet}. \\ &\qquad\qquad (\Phi(\mathrm{Pattern}_2)\ \wedge\ \Phi(\mathrm{where})))) \end{aligned}$$

The translation of $\text{Pattern}_i = \langle E_i, A_i, Pr_i\rangle$ is the formula $\bigwedge r_2(x,y) \wedge \Phi(Pr_i)$ such that $r_2(x,y)$ is the translation of predicate $p = \langle r_1 : C, r_2 : D\rangle$ for every $rel(p,x,y) \in A_i$ with $x : C, y : D$; and $\Phi(Pr_i)$ is the translation of the predicate into Casl. Moreover, the translation of $\text{when} = \langle \text{when}_c, \text{when}_r\rangle$ (or where) is the formula $\bigwedge Rule(v) \wedge \Phi(\text{when}_c)$ such that $Rule(v)$ is the parametric invocation of the rule $(\text{Rule}, v) \in \text{when}_r$, and $\Phi(\text{when}_c)$ is the translation of the predicate.

In the example, the relation `ClassToTable` is translated into Casl as follows:

```
Top_ClassToTable <=> forall p:Package; s:Schema . PackageToSchema(p,s) =>
   forall c:Class; cn:String . namespace(c,p) /\ kind(c,Persistent) /\
      name(c,cn) => exists t:Table . schema(t,s) /\
         name(t,cn) /\ AttributeToColumn(c,t)
```

The formula states that the relation holds whereas for every package and schema satisfying the relation `PackageToSchema`, if there is a persistent class within that package, there must exists a table in the corresponding schema with the same class name. Moreover, the attributes and columns of both elements must be in the relation `AttributeToColumn`.

Given a $\mathcal{I}^{\mathrm{Q}^+}$ theory $T = \langle \Sigma, \Psi\rangle$, a model M of its translated theory (Σ', E) is translated into a Σ-model $\mathcal{M} = \langle \mathcal{M}_1^{\mathrm{M}}, \mathcal{M}_2^{\mathrm{M}}\rangle$ by constructing disjoint models with an interpretation of elements for each corresponding $\mathcal{I}^{\mathrm{M}^+}$ theory.

5 The Environment in Action

We have implemented a prototype of our environment using the Heterogeneous Tools Set (Hets,[3]). Hets is an open source software providing a general framework for formal methods integration and proof management, based on the Theory of Institutions, as introduced above. Based on this foundation, Hets supports a variety of different logics. More specifically, Hets consists of logic-specific tools for the parsing and static analysis of basic logical theories written in the different involved logics (e.g. our extended Csmof and Qvtr institutions), as well as a logic-independent parsing and static analysis tool for structured theories and theory relations. Proof support for other logics can be obtained by using logic translations defined by comorphisms (e.g. from Csmof to Casl). Our prototype and examples can be downloaded together with the Hets distribution.

Within this prototype, MDE experts can specify model transformations in their domain 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 domain with the logical domains needed for verification.

Our problem is represented as a heterogeneous specification using Casl structuring constructs, with three logics: `CASL`, `CSMOF` and `QVTR`. We also perform logic translations through the `CSMOF2CASL` and `QVTR2CASL` comorphisms. Next, there is an excerpt of the heterogeneous specification of the example.

```
(1)   logic CSMOF
      from QVTR/UML get UML |-> UMLMetamodel
      from QVTR/UML_WMult get UML |-> UMLConstraints

(2)   spec UMLProof = UMLMetamodel
      then %implies UMLConstraints end

(3)   logic QVTR
      from QVTR/uml2rdbms get uml2rdbms |-> QVTTransformation

(4)   logic CASL
      spec ModelTransformation = QVTTransformation with logic QVTR2CASL
      then %implies
        . key_RDBMS_Table
        . Top_PackageToSchema
        . Top_ClassToTable
      end
```

Within the CSMOF logic (1) we create two specifications from standard XMI files with the information of the class metamodel and SW-model in Fig. 1. This implies the creation of a representation of signatures and formulas according to the institution defined in Section 3. Another specification is created (2) by extending UMLMetamodel and stating that UMLConstraints is implied. This means that every formula (multiplicity constraint) in the second specification can be derived, thus there must be a proof of it. This is how the satisfaction relation of the Csmof institution is checked. We also use the QVTR logic (3) to create a specification from a standard .qvt file according to the institution defined in Section 3. The only difference with respect to the QVT standard is that instead of using OCL as the expressions language, we use for now a very simple language containing boolean connectives, constants true and false, term equality, strings and variables. Finally, we move into Casl (using the comorphism QVTR2CASL) for creating another specification (4) in which the translation of key and rule formulas defined in Section 3 are implied by the transformation specification. When a proposition, e.g. Top_ClassToTable, is called from the Casl specification, a proof of the implication must be given. We can also translate our specifications and complement them with other constraints which cannot be stated as formulas of the former institutions, e.g. for stating that there cannot be two Classifiers with the same name in the UMLMetamodel specification. For this purpose we use the CSMOF2CASL comorphism as follows.

```
spec MoreProofs = UMLMetamodel with logic CSMOF2CASL
then %implies
  forall x,y:Classifier; str:String . name(x,str) /\ name(y,str) => x = y
end
```

Once our heterogeneous specification is processed, Hets constructs a development graph in which nodes correspond to specifications, some of them with open proof obligations, and arrows to dependencies between them. We have three proof obligations corresponding to those formulas marked as `%implies` within the specifications. Proof goals can be discharged using a logic-specific calculus, e.g. some prover for Casl in the example. The double arrows are heterogeneous theorem links, meaning that the logic changes along the arrow. In the example this corresponds to the extension of specifications by using the comorphisms. It can be noticed that we can use any other logic within the logics graph of Hets through comorphisms. This improves the proof capabilities of our environment.

5.1 Verification Properties

There are several properties that can be verified, some of them related to the computational nature of transformations and target properties of transformation languages, and other to the modeling nature of transformations [2]. The minimal requirement is conformance, i.e. that the source and target models (resp. the transformation specification) are syntactically well-formed instances of the source and target metamodels (resp. the transformation language). Our framework provides this verification in three parts. During the construction of Csmof and Qvtr theories, parsing and static analysis check whether signatures and formulas are well-formed, and (as we explained before) a SW-model within a signature is a structurally well-formed instance of the metamodel in the same signature, as well as a transformation specification given in a formula is well-formed with respect to the signature containing both source and target metamodels. Multiplicity constraints are verified when proving the satisfaction of Csmof formulas. Finally, non-structural constraints are verified by extending both Csmof and Qvtr specifications using other logics, as Casl in the example. Hets also allows for disproving things using consistency checkers. This provides an additional point of view. In particular, we can check if a set of rules have contradictory conditions which could inhibit its execution.

In most cases a general-purpose logic, as provided by Casl, is enough to cover most of the verification approaches in [2]. The inclusion of OCL as an institution will provide additional support in this sense. However, the verification process may depend on the problem to verify, since it is well-known that there is a "state explosion" problem when using automated checkers. Thus, automatic proofs are not always possible. In Hets it is possible to choose the tool we want to use, e.g. not an automated theorem proving system but an interactive theorem prover.

Verification interests go beyond these kinds of problems. When verifying a model transformation we want to consider its elements as a whole and not individually. In this sense, sometimes the notion of a transformation model is used, i.e. a model composed by the source and target metamodel, the transformation specification and the well-formedness rules. We have a transformation model in a Qvtr theory (`QVTTransformation` in the example) which allows to add other properties by combining elements from the source and target metamodels and SW-models. With this we can state model syntax relations, trying to ensure that certain elements or structures of any input model will be transformed into other elements or structures in the output model. This problem arises when, for example, these relations cannot be inferred by just looking at the individual transformation rules. We can also state model semantics relations, e.g. temporal properties and refinement. Besides further work is needed to evaluate the alternatives, there are languages and tools, as ModalCasl and VSE (based on dynamic logics) commonly used for verifying these kinds of things. We could also be interested in working at another abstraction level, i.e. not considering specific SW-models but only metamodels and the transformation specification. This can be useful, for example, for proving that a transformation guarantees some model syntax relations when transforming any valid source SW-model. The problem here is that we need another institutional representation, e.g. we need to consider an abstract representation of a SW-model instead of a fixed one.

6 Related Work

There are some works that define environments for the comprehensive verification of MDE elements based on a unified mathematical formalism, e.g. in [12] rewriting logic is used to analyze MOF-like and QVT-like elements. Since Hets integrates rewriting logic, we can use it instead of our comorphism into Casl. Nevertheless, since our institution is logic-independent it provides more flexibility for the definition of further specific comorphisms into other logics and languages (e.g. UML). In general, the use of a fixed unified mathematical formalism serving as a unique semantic basis can be quite restrictive. With our approach we can move between formalisms, and use a unified mathematical formalism if necessary (e.g. when transforming the whole specification into Casl).

In [13] 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, comorphisms already preserve the semantics with respect to the satisfaction relation.

There are works representing the semantics of UML class diagrams with first-order logic, as in [14]. Since there are no so many alternatives for this representation, these works have similarities with our representation of extended Csmof into Casl. In [15] the authors explain how class diagrams with OCL constraints can be translated into Casl. However, their definition is informally presented, and not in terms of a comorphism. In [16] the authors define a comorphism from UML class diagrams with rigidity constraints to ModalCASL (an extension of Casl). Since our $\mathcal{I}^{\mathrm{M}^+}$ institution is an adaptation of the institution for UML class diagrams, the comorphisms have some aspects in common, as the translation of formulas, but without the modal logic particularities.

There are several approaches to heterogeneous specification for traditional software development, but there is little tool support. CafeOBJ [17] is an institution-based approach providing a fixed cube of eight logics and twelve projections (formalized as institution morphisms), not allowing logic encodings (formalized as comorphisms). Thus, it is not an option for the definition of our environment.

Moreover, HeteroGenius [18] is another institution-based framework allowing the interaction between different external tools for performing hybrid analysis of a specification. However, the framework is not formally defined or available to be used as a basis for our environment.

7 Conclusions and Future Work

We have presented the implementation of an environment for the formal verification of different MDE aspects using heterogeneous verification approaches, which is based on the ideas introduced in [4]. The environment was integrated into Hets by defining comorphisms from institutions representing MDE elements to Casl, the core language of Hets. The existent connections between Casl and other logics within Hets broadens the spectrum of logical domains in which the verification of MDE elements can be addressed. The environment supports a separation of duties between software developers (MDE and formal methods experts) such that a formal perspective is available whenever it is required. A developer imports MDE elements, supplement them with verification properties specified using any other logic supported by Hets, and perform the heterogeneous verification assisted by the tool. Since the implementation can generate a heterogeneous specification from the same files used by MDE practitioners, and there is no need of rewriting MDE building block in each logic involved, the environment is scalable without human assistance.

A current drawback is the inexistence of an institution for OCL, in which QVT is strongly based. For now we have considered a very simple expressions language, but the definition of such institution is subject of further work. In the same sense, we expect to extend the institutions to include some elements not considered before. This will strengthen the formal environment for MDE. Since our institutions formalize languages strongly related with those in the UML ecosystem, it will be interesting to explore the possibility of integrating them with other languages, as those already defined as institutions in [19].

We need to continue bridging the gap between MDE and formal verification in terms of tool development in order to practitioners really be able to benefit from our approach. We can connect the definition of the MDE elements in any popular tool with an automatic generation of the heterogeneous specification. Moreover, we could perform an automated verification of some properties (if possible) by running Hets in background. We also need to improve feedback from existing formal tools, which needs better traceability between the problem definition and the results given by a verification tool. We can define some traceability links from comorphisms, interpret the output of the verification tool and return something that the MDE practitioner can interpret. Moreover, the environment deals with many verification properties, but a deeper understanding of this (as for example about the behavior of models) is a must. In this sense, we can use the knowledge in [2] to provide a guide for the selection of the "right" verification approach for the problem which is of interest to verify. We also need to apply our approach to industrial, real-size examples for strengthening the results.

References

1. Kent, S.: Model driven engineering. In: Butler, M., Petre, L., Sere, K. (eds.) IFM 2002. LNCS, vol. 2335, pp. 286–298. Springer, Heidelberg (2002)
2. Calegari, D., Szasz, N.: Verification of model transformations: A survey of the state-of-the-art. ENTCS **292**, 5–25 (2013)
3. Mossakowski, T.: Heterogeneous specification and the heterogeneous tool set. Technical report, Universitaet Bremen, Habilitation thesis (2005)
4. Calegari, D., Szasz, N.: Institution-based semantics for MOF and QVT-relations. In: Iyoda, J., de Moura, L. (eds.) SBMF 2013. LNCS, vol. 8195, pp. 34–50. Springer, Heidelberg (2013)
5. Goguen, J.A., Burstall, R.M.: Institutions: Abstract model theory for specification and programming. J. ACM **39**, 95–146 (1992)
6. Mossakowski, T., Haxthausen, A.E., Sannella, D., Tarlecki, A.: Casl - the common algebraic specification language: Semantics and proof theory. Computers and Artificial Intelligence **22**, 285–321 (2003)
7. OMG: Meta Object Facility (MOF) 2.0 Core Specification. Specification Version 2.0, Object Management Group (2003)
8. OMG: Object Constraint Language. Formal Specification Version 2.2, Object Management Group (2010)
9. OMG: Meta Object Facility (MOF) 2.0 Query/View/Transformation. Final Adopted Specification Version 1.1, Object Management Group (2009)
10. Calegari, D.: Heterogeneous Verification of Model Transformations. PhD thesis, Universidad de la República - PEDECIBA (2014). https://www.fing.edu.uy/inco/pedeciba/bibliote/tesis/tesisd-calegari.pdf
11. Codescu, M.: Generalized theoroidal institution comorphisms. In: Corradini, A., Montanari, U. (eds.) WADT 2008. LNCS, vol. 5486, pp. 88–101. Springer, Heidelberg (2009)
12. Boronat, A., Heckel, R., Meseguer, J.: Rewriting logic semantics and verification of model transformations. In: Chechik, M., Wirsing, M. (eds.) FASE 2009. LNCS, vol. 5503, pp. 18–33. Springer, Heidelberg (2009)
13. Lano, K., Rahimi, S.K.: Model transformation specification and design. Advances in Computers **85**, 123–163 (2012)
14. Shan, L., Zhu, H.: Semantics of metamodels in UML. In: Chin, W., Qin, S. (eds.) TASE 2009, pp. 55–62. IEEE Computer Society (2009)
15. Bidoit, M., Hennicker, R., Tort, F., Wirsing, M.: Correct realizations of interface constraints with OCL. In: France, R., Rumpe, B. (eds.) UML 1999. LNCS, vol. 1723, pp. 399–415. Springer, Heidelberg (1999)
16. James, P., Knapp, A., Mossakowski, T., Roggenbach, M.: Designing domain specific languages – a craftsman's approach for the railway domain using Casl. In: Martí-Oliet, N., Palomino, M. (eds.) WADT 2012. LNCS, vol. 7841, pp. 178–194. Springer, Heidelberg (2013)
17. Diaconescu, R., Futatsugi, K.: Logical foundations of CafeOBJ. Theor. Comput. Sci. **285**, 289–318 (2002)
18. Giménez, M., Moscato, M.M., Pombo, C.G.L., Frias, M.F.: Heterogenius: a framework for hybrid analysis of heterogeneous software specifications. In: Aguirre, N., Ribeiro, L. (eds.) LAFM 2013. EPTCS, Vol. 139, pp. 65–70 (2014)
19. Cengarle, M.V., Knapp, A., Tarlecki, A., Wirsing, M.: A heterogeneous approach to UML semantics. In: Degano, P., De Nicola, R., Meseguer, J. (eds.) Montanari Fest. LNCS, vol. 5065, pp. 383–402. Springer, Heidelberg (2008)