HAL
open science

A formal framework to prove the correctness of model driven engineering composition operators

Mounira Kezadri, Marc Pantel, Benoit Combemale, Xavier Thirioux

To cite this version:

Mounira Kezadri, Marc Pantel, Benoit Combemale, Xavier Thirioux. A formal framework to prove the correctness of model driven engineering composition operators. ICFEM'14 - 16TH INTERNATIONAL CONFERENCE ON FORMAL ENGINEERING METHODS, Nov 2014, Luxembourg, Luxembourg. hal-01024067

HAL Id: hal-01024067

https://inria.hal.science/hal-01024067

Submitted on 15 Jul 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A formal framework to prove the correctness of model driven engineering composition operators

Mounira Kezadri Hamiaz[1], Marc Pantel[1], Benoit Combemale[2], and Xavier Thirioux[1]

[1] Université de Toulouse, IRIT, France
[2] Université de Rennes 1, IRISA, France

Abstract. Current trends in system engineering combine modeling, composition and verification technologies in order to harness their ever growing complexity. Each composition operator dedicated to a different modeling concern should be proven to be property preserving at assembly time. These proofs are usually burdensome with repetitive aspects. Our work[3] targets the factorisation of these aspects relying on primitive generic composition operators used to express more sophisticated language specific ones. These operators are defined for languages expressed with `OMG MOF` metamodeling technologies. The proofs are done with the Coq proof assistant relying on the Coq4MDE framework defined previously. These basic operators, `Union` and `Substitution`, are illustrated using the `MOF` Package Merge as a composition operator and the preservation of model conformance as a verified property.

1 Introduction and motivation

Safety critical systems are getting more and more complex and software intensive while the safety rules are more and more stringent (e.g. DO-178 in aeronautics [41]). Several technologies are playing a key role to tackle these issues.

First, Model-Based Systems Engineering (MBSE) relying on Model Driven Engineering (MDE) [7] promotes the use of models at the various development phases, composition operators and model transformations to automate parts of the development. Models are abstract specifications of the various system concerns/aspects that are usually purpose oriented and allow early Validation and Verification (V & V) (e.g. the new DO-331 standard [40]).

Then, formal methods allow the assessment of the completeness and consistency of specification models, and of the correctness of design models and implementations with respect to specification models. Their mathematical nature provides high level of confidence in their result (e.g. the new DO-333 standard [39]).

In order to benefit from these technologies and avoid doing all the V & V activities on the final system, safety standards require the associated process,

[3] This work was partly funded by the French ministry of research through the ANR-12-INSE-0011 grant for the GEMOC project.

methods and tools to be qualified (e.g. the new DO-330 standard: Software Tool Qualification Considerations that adapts the DO-178C to the development and the verification tools [42]). These qualification activities are very costly and can benefit from the use of formal methods relying on the DO-333 standard [39].

To ease the integration of formal specification and verification technologies, some of the authors proposed in [44] a formal embedding of some key aspects of `MDE` in Set Theory. This embedding was then implemented using the Calculus of Inductive Construction [15] and the Coq[4] proof-assistant. This framework called Coq4MDE[5] provides sound mathematical foundations for the study and the validation of `MDE` technologies. The choice of constructive logic with type theory as formal specification language allows to extract prototype tools from the executable specification that can be used to validate the specification itself with respect to external tools implementing the `MDE` principles (for example, in the Eclipse[6] Modeling Project).

We proposed in [25] an extension of Coq4MDE to support the Invasive Software Composition (`ISC`) [1] style. We then experimented the design of formalized primitive operators and their use to ease the implementation, and especially the proof of correctness of the `ISC` operators and other high level ones. This contribution specifies and assesses the properties of the primitive composition operators `Union` and `Substitution` that can be used to specify higher level composition operators sharing parts of their implementation and associated properties. The assessed property is the conformance of models to metamodels which is mandatory for all model composition operators. Our proposal is illustrated by the use of these primitive operators to specify and prove the `MOF` (Meta Object Facility) [32] Package Merge. Other sophisticated composition operators like `ISC` (adaptation and glueing) [1] [20] and aspect weaving [29] can also be built from these primitive ones, and their proofs of conformance preservation are also built from the ones of primitive operators. One key point is the use of property specific contracts (pre and post conditions) for the composition operators in order to ensure compositional verification.

This contribution is structured as follows. First, the concepts and an example for a targeted high level operator and the expected properties are given in Section 2. Then, Section 3 presents the formal support for the Coq4MDE framework that is extended to handle the definition of the primitive composition operators associated with the proofs of property preservation. Then, some use cases are provided in Section 4. Related work are given in Section 5. Finally, conclusion and perspectives are provided in Section 6.

2 Use case : MOF Package Merge

This section introduces our verification proposal applied to the `MOF` Package Merge operator.

[4] `http://coq.inria.fr`
[5] `http://coq4mde.enseeiht.fr/FormalMDE/`
[6] `https://www.eclipse.org/`

2.1 Model Driven Engineering

The core principle of MDE is *"everything is a model"* [7]. Models are defined using modeling languages. Metamodels are models of modeling languages defined using metamodeling languages. A model M is conforming to a metamodel MM if MM models the language used to define M. Metamodels, like data types, define the structure common to all its conforming models, but can also give semantics properties like dependent types. Derived from [23], Coq4MDE separates the model level (value or object) from the metamodel level (type or class), and describes them in Coq with different types.

2.2 Meta-Object Facility

The OMG has standardized the MOF, that provides a reflexive metamodeling language (i.e. MOF is defined as a model in the MOF language). MOF is used for the specification of the OMG modeling language standards like MOF itself, UML [33], OCL [46], SysML [21] and many others. The relation between MOF and the metamodels is the same as the one between a metamodel and its conforming models. Based on these principles, the OMG introduced the MDA (Model Driven Architecture) [6] view of software modeling illustrated by the pyramid given in Figure 2. Since the MOF version released in 2006 [31], a kernel named EMOF was extracted from the complete version of MOF (CMOF). EMOF provides a minimal set of elements required to model languages. Figure 1 gives the key concepts of EMOF specified as an UML class diagram. The principal concept is Class to define classes (usually called metaclasses) that represent concepts in a modeling language. Classes allow to create objects in models. The type of an object is the class that was used to create it. Classes are composed of an arbitrary number of Property (we will call them reference and attribute in order to avoid ambiguities with the model property we want to assess) and Operation (not detailled here). References allow to create the relations between the objects in the models. Classes can inherit references and attributes from other classes. Inheritance is expressed using the superClass reference from Class. Inheritance introduces a subtyping relation between the types associated to classes. Classes can be abstract (isAbstract): no object can have the type associated to an abstract class as smallest type according to the subtyping relation. Property has a lower and upper attributes that bound the number of objects contained in a given reference. Two references can be opposite to build a bidirectional relation between objects in a model.

In MBSE, many models are used to represent the various system's concerns. They must be composed to build the global system. We present in the following subsection the MOF composition operator that is used in the OMG UML specification [33] to define and assemble metamodel parts.

2.3 MOF Package Merge

Package Merge is a directed relation from a package (merged) to another package (receiving) as mentioned in Figure 3. It can be seen as an operation that takes

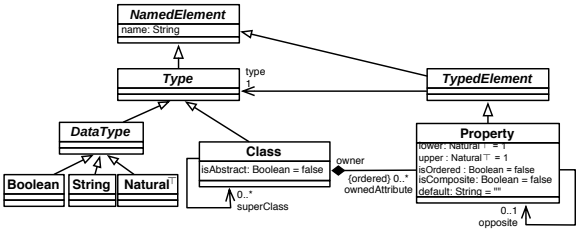


Fig. 1. The basic concepts of EMOF

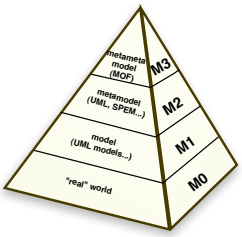


Fig. 2. The OMG Pyramid

the content of both packages and produces a new package (`resulting`) that merges the contents of the initial packages. In the case where some elements in these packages represent the same entity, theirs contents must be combined according to the rules given in [33].

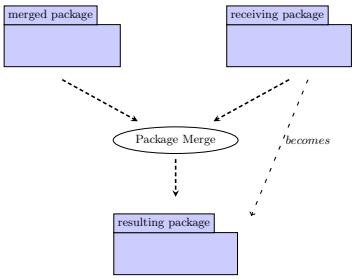


Fig. 3. Conceptual view of the Package Merge [33]

When there are no conflicts between the two packages, Package Merge is equivalent to an union for all the packages elements (in fact, a special union that preserves the references and attributes for classes).

When conflicts occur, two major kinds are considered: a) when the same class has different attributes and references in the two packages, this conflict is resolved by keeping all the attributes and references from both classes in the `merged` and `receiving` models (this is obviously an application of the previous special union operator), b) when two corresponding attributes or references have different values (for example different multiplicity values); the conflicts are resolved by combining the values according to the semantics of the conflicting attribute or reference.

2.4 Expected property

The expected property in this use case is the preservation of the metamodel conformance during composition. For a model M and a metamodel MM, this property checks that: 1) every object o in M was created from a class C in MM. 2) every relation between two objects in M is such that there exists, in MM, a reference between the two classes used for creating the two objects. 3)

every semantics property defined in MM is satisfied in M. The semantics properties from `EMOF` (see Figure 1) are: Inheritance (`subClass`), Abstract classes (`isAbstract`), Multiplicities (`lower`, `upper`), Opposite (`isOpposite`) and Composite (`areComposite`) references.

As verifying these properties directly for the `MOF` Package Merge operator is complex and contains many common aspects with other high level composition operators, we target a divide and conquer approach to capture these commonalities.

2.5 Verification strategy

We advocate the use of generic primitive composition operators that are then used to specify and prove more sophisticated ones like `MOF` Package Merge or `ISC`. We target pragmatic compositional verification: minimize the residual verification that must be conducted on the result of the composition of correct models. We rely on a simple methodology to design the contract (pre and post conditions) for the composition operators in that purpose. If Φ is the expected property for a model built using composition operators, then Φ must be, on the one hand, the postcondition on the model resulting from the application of each operator; and, on the other hand, the weakest precondition on each parameter of each operator. These preconditions are eventually consolidated with an additional glueing property Ψ depending on the value of all the parameters of the operator. Ψ is the residual property that must be checked at each composition.

Definition 1 (Correct composition operator). *For a set of models $m_1, \ldots, m_n$ and an n-ary composition operator f over models, we say f is correct (or property preserving) with respect to a property Φ and a glueing condition Ψ if:*

$$\bigwedge_{1 \leq i \leq n} \Phi(m_i) \wedge \Psi(m_1, \ldots, m_n) \Rightarrow \Phi(f(m_1, \ldots, m_n))$$

The verification that the composition operators are correct will be conducted using the Coq proof assistant, the details are presented in the next Section.

3 Formalization using Coq4MDE

The concepts of Model and Metamodel are formally defined in Coq4MDE in the following way. Let us consider two sets of labels: `Classes`, respectively `References`, represents the set of all possible classes, respectively reference, labels. Then, let us consider instances of such classes, the set `Objects` of object labels. `References` includes a specific *inh* label used to specify the direct inheritance relation.

Definition 2 (Model).
Let $\mathscr{C} \subseteq$ `Classes` be a set of class labels. Let $\mathscr{R} \subseteq \{\langle c_1, r, c_2 \rangle \mid c_1, c_2 \in \mathscr{C}, r \in$ `References`$\}$[7] be a set of references between classes.

[7] $\langle c_1, c_2, r \rangle$ in the Coq code is denoted here for simplification as: $\langle c_1, r, c_2 \rangle$.

A `Model` *over* $\mathscr{C}$ *and* $\mathscr{R}$*, written* $\langle MV, ME\rangle \in Model(\mathscr{C}, \mathscr{R})$ *is a multigraph built over a finite set* MV *of typed object vertices and a finite set* ME[8] *of reference edges. such that:*

$$MV \subseteq \{\langle o, c\rangle \mid o \in \mathit{Objects}, c \in \mathscr{C}\}$$
$$ME \subseteq \{\, \langle\langle o_1, c_1\rangle, r, \langle o_2, c_2\rangle\rangle \,|\, \langle o_1, c_1\rangle, \langle o_2, c_2\rangle \in MV, \langle c_1, r, c_2\rangle \in \mathscr{R} \,\}$$

In case of inheritance, the same object label will be used several times in the same model graph. It will be associated to the different classes in the inheritance hierarchy going from a root down to the class used to create the object. This label reuse encodes subtyping. Direct inheritance is represented in the metamodel with a special reference called *inh*. The following property states that c_2 is a direct subclass of c_1.

$$subClass(c_1, c_2 \in \mathtt{Classes}, \langle MV, ME\rangle) \triangleq \forall o \in \mathtt{Objects},$$
$$\langle o, c_2\rangle \in MV \Rightarrow \langle\langle o, c_2\rangle, inh, \langle o, c_1\rangle\rangle \in ME$$

Abstract Classes that are specified in a metamodel using the *isAbstract* attribute are not suitable for instantiation. They are often used to represent abstract concepts that gather common attributes and references.

$$isAbstract(c_1 \in \mathtt{Classes}, \langle MV, ME\rangle) \triangleq \forall o \in \mathtt{Objects},$$
$$\langle o, c_1\rangle \in MV \Rightarrow \exists c_2 \in \mathtt{Classes}, \langle\langle o, c_2\rangle, inh, \langle o, c_1\rangle\rangle \in ME$$

Definition 3 (Metamodel). *A* `MetaModel` *is a multigraph representing classes as vertices and references as edges as well as semantics properties over instantiation of classes and references. It is represented as a pair composed of a multigraph* (MMV, MME) *built over a finite set* MMV *of vertices and a finite set* MME *of edges, and as a predicate (*$conformsTo$*) over models representing the semantics properties.*

A `MetaModel` *is a pair* $\langle (MMV, MME), conformsTo\rangle$ *such that:*

$$MMV \subseteq \mathit{Classes}$$
$$MME \subseteq \{\langle c_1, r, c_2\rangle \mid c_1, c_2 \in MMV, r \in \mathit{References}\}$$
$$conformsTo : Model(MMV, MME) \rightarrow Bool$$

A minimum and maximum number of values that can be associated to an attribute or reference can be defined using the *lower* and *upper* attributes. This pair is usually named *multiplicity*. In order to ease the manipulation of this datatype, we introduce the type $\text{Natural}^{\top} = \mathbb{N} \cup \{\top\}$. Using both attributes, it is used to represent a range of possible numbers of values. Unbounded ranges can be modelled using the $\top$ value for the *upper* attribute.

$$lower(\mathtt{c_1} \in \mathtt{MMV}, \mathtt{r_1} \in \mathtt{MME}, \mathtt{n} \in Natural^{\top}, \langle \mathtt{MV}, \mathtt{ME}\rangle) \triangleq$$
$$\forall\ o \in \mathtt{Objects}, \langle \mathtt{o}, \mathtt{c_1}\rangle \in \mathtt{MV} \Rightarrow |\{\mathtt{m_2} \in \mathtt{MV} \mid \langle\langle \mathtt{o}, \mathtt{c_1}\rangle, \mathtt{r_1}, \mathtt{m_2}\rangle \in \mathtt{ME}\}| \geq \mathtt{n}$$

An analogous formalization is defined for the `upper` property replacing $\geq$ by $\leq$.

[8] $\langle\langle o_1, c_1\rangle, r, \langle o_2, c_2\rangle\rangle$ is denoted in the Coq code as: $\langle\langle o_1, c_1\rangle, \langle o_2, c_2\rangle, r\rangle\rangle$.

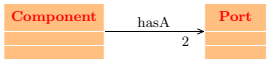
A reference can be associated to an *opposite* reference. It means that, in a model conforming to its metamodel, for each link instance of this reference between two objects, there must exists a link in the opposite direction between these two objects.

$$\begin{aligned} &isOpposite(\mathtt{r}_1, \mathtt{r}_2 \in \mathtt{MME}, \langle \mathtt{MV}, \mathtt{ME} \rangle) \triangleq \\ &\quad \forall\, \mathtt{m}_1, \mathtt{m}_2 \in \mathtt{MV}, \langle \mathtt{m}_1, \mathtt{r}_1, \mathtt{m}_2 \rangle \in \mathtt{ME} \Leftrightarrow \langle \mathtt{m}_2, \mathtt{r}_2, \mathtt{m}_1 \rangle \in \mathtt{ME} \end{aligned}$$

A reference can be *composite*. In this case, instances of the target concept belong to a single instance of source concepts.

$$\begin{aligned} &areComposite(\mathtt{c}_1 \in \mathtt{MMV}, R \subseteq \mathtt{MME}, \langle \mathtt{MV}, \mathtt{ME} \rangle) \triangleq \\ &\forall\, o \in \mathtt{Objects} \Rightarrow \quad |\{\mathtt{m}_1 \in \mathtt{MV} \mid \langle \mathtt{m}_1, \mathtt{r}, \langle \mathtt{o}, \mathtt{c}_1 \rangle \rangle \in \mathtt{ME}, \mathtt{r} \in R\}| \leq 1 \end{aligned}$$

Figure 4 shows a simple example of metamodel on the left and his Coq4MDE representation on the right.

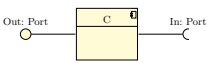


$$\begin{aligned} &MMV = \{Component, Port\}, \\ &MME = \{(Component, hasA, Port)\}, \\ &conformsTo = lower(Component, hasA, 2) \\ &\qquad \wedge\, upper(Component, hasA, 2)\rangle \end{aligned}$$

Fig. 4. A metamodel in the Coq4MDE notation

Given one `Model` M and one `MetaModel` MM, we can check conformance using the $conformsTo$ predicate embedded in MM. This predicate identifies the set of models conforming to a metamodel.

Figure 5 shows an example of a model that conforms to the metamodel given in Figure 4. At the right of this figure is the Coq4MDE representation associated to this model. All the structural (well typedness) and semantics properties of the metamodel in Figure 4 are respected in this model. Namely, C is an object of the class $Component$, In and Out are objects of the class $Port$, the component C is linked with the relation $hasA$ to exactly 2 ports (the *lower* and *upper* attributes specified in the metamodel are respected).



$$\begin{aligned} &MV = \{(C, Component), (In, Port), (Out, Port)\}, \\ &ME = \{((C, Component), hasA, (In, Port)), \\ &\qquad ((C, Component), hasA, (Out, Port))\} \end{aligned}$$

Fig. 5. A model in the Coq4MDE notation

For the implementation, a model in Coq4MDE is a multigraph defined as finite sets of vertices and edges satisfying some properties. Based on this model's definition, every new operator in the Coq4MDE framework needs to be addressed at the three hierarchically related levels (the sets level, the graph level and the `MDE` level). The following `Union` and `Substitution` operators definitions follow this three levels schema.

3.1 The Union primitive operator

For the set level, we use the union encoded in the Coq library `Uniset`[9] that we note $\cup$. The same notation or symbolic abbreviation $\cup$ is used in our Coq code.

[9] `http://coq.inria.fr/stdlib/Coq.Sets.Uniset.html`

For the graph's level, the result is defined with a proof[10] by induction that the union of two graphs is also a graph. The vertices/edges set for the resulting graph is the union of vertices/edges sets of the two initial graphs.

The union of two models $\langle vs_1, es_1 \rangle$ and $\langle vs_2, es_2 \rangle$ is the union of their vertices and edges sets in addition to the proof that the two sets constitute a graph. For simplification, this can be denoted as follows: $\langle vs_1 \cup vs_2, es_1 \cup es_2 \rangle$.

3.2 The Substitution primitive operator

As explained in Definition 1, the models are graphs having as nodes typed objects. For example, (o, c) is a model's object whose type is c and whose name is o. The `Substitution` operator aims to replace the name of an object of a model by another name. This operator is also defined using the three hierarchical levels.

Substituting a model's object whose name is *src* by a model's object whose name is *dst* inside the sets of vertices and edges is implemented using a generic *map* operator encoded in Coq using three basic functions: *mapv*, *mapa* and *mape* that are applied respectively on: the model's objects, the references and the edges. The function *mapv* is defined as follows: $mapv\ (o, c) = \begin{cases} (dst, c)\ if\ \ o = src \\ (o, c)\ otherwise. \end{cases}$
The function *mape* replaces the names of the model's objects in the edges such as: $mape\ (v_1, a, v_2) = (mapv\ v_1, mapa\ a, mapv\ v_2)$.

The graph's image[11] is then constructed from the initial graph using the images of the vertices and the edges by *mapv*, *mapa* and *mape*.

The sets and the graph obtained from the previous levels and the associated proofs constitute the substituted model.

3.3 Proofs of primitive operators

We consider the well typedness ($instanceOf$) and the semantics properties discussed in Section 2.4. For every property (Φ), we prove the preservation by the primitive operators. Some properties are fully compositional: no residual verification activity (Ψ) needs to be conducted at the composition time. Others will require additional verification activities (Ψ) that can be modeled as preconditions.

For space reason, we only give the example of the hierarchy property (Φ: subClass) for the `Union` operator. Theorem 1 states[12] that the hierarchy property is preserved by `Union`. So, for any classes c_1 and c_2, if c_1 is a *subClass* of c_2 in the models M_1 and M_2, then c_1 is also a *subClass* of c_2 in the model resulting from the `Union` of M_1 and M_2.

Theorem 1. (subClassUnionPreserved) $\forall\ \mathtt{M_1}\ \mathtt{M_2} \in Model,\ c_1\ c_2 \in \mathtt{Classes},$ $subClass(c_1\ c_2\ \mathtt{M_1})\ \wedge\ subClass(c_1\ c_2\ \mathtt{M_2}) \Rightarrow subClass(c_1\ c_2\ (Union\ \mathtt{M_1}\ \mathtt{M_2}))$.

[10] `http://coq4mde.enseeiht.fr/FormalMDE/Graph.html#MG.EG`
[11] `http://coq4mde.enseeiht.fr/FormalMDE/Subst_Verif.html\#elements`
[12] `http://coq4mde.enseeiht.fr/FormalMDE/Union.html#SCUP`

Table 1 summarizes the pre and postconditions for the verification of the metamodel conformance for the `Union` and `Substitution` operators. The proofs for the `Union`, respectively `Substitution`, operator are accessible at: `http://coq4mde.enseeiht.fr/FormalMDE/Union_Verif.html`, respectively `http://coq4mde.enseeiht.fr/FormalMDE/Subst_Verif.html`.

Φ	M_r=Substitution $((\mathtt{o}_1, \mathtt{c}_1), (\mathtt{o}_2, \mathtt{c}_2), \mathrm{M})$	M_r=Union $(M_1,\ M_2)$
$instanceOf$	$\Psi(M)$=True	$\Psi(M_{i\in\{1,2\}})$=True
$subClass$	$\Psi(M)$=True $\Phi(M_r) = subClass(c_1, c_2, M)$	$\Psi(M_{i\in\{1,2\}})$=True $\Phi(M_r) = subClass(c_1, c_2, M_r)$
$isAbstract$	$\Psi(M)$=True	$\Psi(M_{i\in\{1,2\}})$=True
$lower$	$\Psi(M) = (\mathtt{c}_1 = \mathtt{c}_2) \wedge ((o_2, c) \notin \mathtt{MV})$	$\Psi(M)$=$lowerCond(c, r, nM_1, M_2)$
$upper$	$\Psi(M) = (\mathtt{c}_1 = \mathtt{c}_2) \wedge ((o_2, c) \notin \mathtt{MV})$	$upperCond(c, r, n, M_1, M_2)$
$isOpposite$	$\Psi(M)$=True	$\Psi(M_{i\in\{1,2\}})$=True
$areComposite$	$\Psi(M) = (\mathtt{c}_1 = \mathtt{c}_2) \wedge ((o_2, c) \notin \mathtt{MV})$	$\Psi(M) = 1 >$ $\lvert\{\mathtt{o}_2 \in \mathtt{MV}_1 \mid \langle\langle \mathtt{o}, \mathtt{c}\rangle, \mathtt{r}, \mathtt{o}_2\rangle \in \mathtt{ME}_1\}\rvert$ $+\lvert\{\mathtt{o}_2 \in \mathtt{MV}_2 \mid \langle\langle \mathtt{o}, \mathtt{c}\rangle, \mathtt{r}, \mathtt{o}_2\rangle \in \mathtt{ME}_2\}\rvert$ $-\lvert\{\mathtt{o}_2 \in (\mathtt{MV}_1 \cap \mathtt{MV}_2) \mid$ $\langle\langle \mathtt{o}, \mathtt{c}\rangle, \mathtt{r}, \mathtt{o}_2\rangle \in (\mathtt{ME}_1 \cap \mathtt{ME}_2)\}\rvert$

Table 1. pre and postconditions for the Union and Substitution operators

The basic Coq4MDE framework is about 1107 lines, the actual version containing the primitive composition operators and also the proved implementation of the Package Merge described in the next section is about 18000 lines with about 300 Lemmas and Theorems and 200 Definitions. The proofs for the elementary operators are about 3300 lines. The implementation and proofs of the Package Merge using the elementary operators is about 7200, this implementation take advantage from reusing the proofs previously done for the primitive operators. The alternative is the implementation without elementary operators and that would require multiple repetitions of the elementary proofs and would be about 20400 lines. So, our approach enables a reduction with more then 180% in this case.

4 Validation

The primitive operators have been used for the implementation of higher level composition operators including `MOF` Package Merge, `ISC` (considering the adaptation and glueing of components) and aspect weaving. These operators share parts of their implementation and conformance preservation proof that can be captured by the use of the primitive operators. For their implementations, other model operations are required (e.g. extraction of matching between models, verification of some conditions, ...) but the only modifications of the models are primitive substitutions and unions. The verifications of the fully compositional properties reuse directly the proofs of the primitive operators without any additional parts. The verification of the properties requiring additional preconditions needs to ensure that the preconditions are satisfied.

We show mainly in this section that our minimal set of primitive operators is sufficient to formalize a high level operator like the `MOF` Package Merge. A mature formalization for the `ISC` operators based on the elementary ones is also available, for the details see [24].

The Package Merge implementation as summarized hereafter is accessible at: `http://coq4mde.enseeiht.fr/PackageMergeCoq/`.

To illustrate our methodology, we give an example derived from [48].

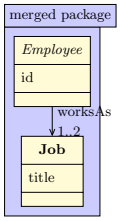


Fig. 6. `BasicEmployee`

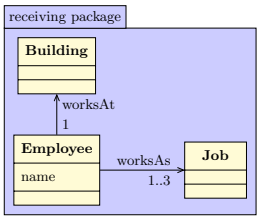


Fig. 7. `EmployeeLocation`

The source package (in this case the package `BasicEmployee` mentioned in Figure 6) is the package `receiving`. The package `EmployeeLocation` shown in Figure 7 is the package `merged`. This package contains the additional elements that must be merged with the package `receiving`. Two conflicts occur between the models `merged` and `receiving`. The first one is related to the attribute *upper* of *worksAs* (the maximal bound is equal to 2 in the model `BasicEmployee` (Figure 6) and is equal to 3 in the model `EmployeeLocation` (Figure 7)). The second conflict is related to the class *Employee* that is abstract (name in italic) in the `merged package` and concrete in the `receiving package`.

The resolution of this kind of conflicts is done according the the `UML` specification [33]. The rule to resolve the conflict for the `upper` attribute is: $upper_{Resulting} = max(upper_{Merged}, upper_{Receiving})$. The rule for the `isAbstract` attribute is: $isAbstract_{Resulting} = isAbstract_{Merged} \wedge isAbstract_{Receiving}$. The list of all the possible transformations is available on page 166 of the specification [33].

Concretely in our abstract syntax, we manipulate the metamodels as models conforming to `MOF`, so the abstraction property for classes is represented with attributes *isAbstract* suffixed with the name of the class (this attribute is equal to True in the model `BasicEmployee` (Figure 6) and equal to False in the model `EmployeeLocation` (Figure 7)). The same principle is used to represent all the properties linked to `MOF` such as `lower` and `upper`. We show in Figure 8 the representation of the package `BasicEmployee` as a model conforming to `MOF`. The `EmployeeLocation` package is represented using the same principle, we don't show it here for space reason. The first step is to resolve all the conflicts. For this, the `Substitution` operator is applied twice. The first application replaces 2 by 3 for the $upper_{Job}$ attribute in the `merged` model. The second application of the `Substitution` operator replaces $True$ by $False$ for the $isAbstract_{Employee}$ attribute in the `merged` model.

Once the conflicts are resolved, the final step is the `Union` of the obtained models `merged` and `receiving` (the constraints of the `Union` operator are sat-

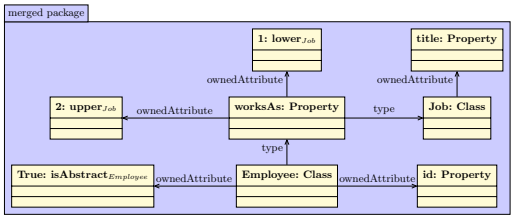


Fig. 8. An excerpt from the `BasicEmployee` model

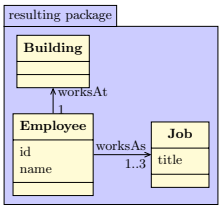


Fig. 9. The resulting metamodel

isfied in this case). The result is exactly the merge of the two packages `merged` and `receiving` shown in Figure 9.

In the previous example, the `Package Merge` is expressed using the primitive composition operators `Union` and `Substitution`. Defining the `Package Merge` in this manner ensures that the resulting model is well typed in relation with the packages `merged` and `receiving` and also that it satisfies the semantics properties of the metamodel when the preconditions are satisfied.

We have also experimented the use of our primitive operators to define: the aspect weaving [29], the merge of statecharts specifications [30], the weave of State and Sequence Diagrams, the attributes composition [43] and also to reimplement the operators of the Invasive Software Composition (`ISC`) [1].

5 Related work

This work targets a formal certified model composition framework. Our notion of model follows the `MDE` vision and we are interested in the problems of composition and compositional verification. This work is related to several issues highlighted in this Section. First, we take a look at some composition approaches and we rely on a `MOF` Package Merge formalization to explain our contribution. Then, we discuss some formalizations of the `MDE` to position our proposal. Finally, we present some work on compositional verification to situate our work.

5.1 Composition approaches

We previously proposed in [25] [19] the formalization and verification of some ReuseWare [20] operators. Several composition methods have focused on the implementation of the merge operators using mappings between models like Rational Software Architect[13], Bernstein et al. data model [6], Atlas Model Weaver[14] [16], Epsilon[15], `Theme/UML` [13] and `EMF` Facet[16]. The ReuseWare operators and the `MOF` Package Merge operators as presented in the previous

[13] `http://www-306.ibm.com/software/awdtools/architect/swarchitect/`
[14] `http://www.eclipse.org/gmt/amw/`
[15] `http://www.eclipse.org/gmt/epsilon/`
[16] `www.eclipse.org/proposals/emf-facet/`

frameworks and also in [11] and [4] are defined as a composition of the primitive operators. The advantages of these implementation relying are the proof of termination of the composition operators (ensured by the Coq proof-assistant), the properties verification for the composition and the support for the extraction of the validated executable code.

Zito in [48] presented an implementation using `Alloy`[17] [22] of the `MOF Package Merge`. As `Alloy` has a poor performance when analysing models with many signatures (e.g. 20 signatures or higher), the analysis is currently limited to an interval between 5 and 10. The `UML` metamodel contains more than 30 classes which are all modelled with a signature. The authors applies several strategies to reduce the size of the `Alloy` model: reduce the depth of inheritance, merge similar classes, do not model classes that do not contain any information and do not change by applying the fusion (e.g. PackageImport, PackageMerge, ElementImport, Comment and Constraint), merge multiple inheritance, eliminate recursion, and finally do not model derived attributes and associations.

Elaborating a proved development allows specifying generically the properties and verifying these properties a priori for all the instances. This universal quantification used in the proofs enables to avoid all the constraints and the limitation in relation with the models' size.

5.2 Formalization of MDE

A lot of work was conducted aiming to formalize the concepts of `MDE`.

First, MoMENT (MOdel manageMENT) [8] is an algebraic model management framework that provides a set of generic operators to manipulate models. The metamodels are represented as algebraic specifications and the operators are defined independently of the metamodel using the Maude language [14].

Also, A. Vallecillo et al. have designed and implemented a different embedding of metamodels, models ([38]) and model transformations ([45]) using Maude. This embedding is shallow, it relies strongly on the object structure proposed by Maude in order to define model elements as objects, and relies on the object rewriting semantics in order to implement model transformations.

Furthermore, I. Poernomo has proposed an encoding of metamodels and models using type theory ([35]) in order to allow correct by construction development of model transformation using proof-assistants like Coq ([36]). Some simple experiments have been conducted using Coq mainly on tree-shaped models ([37]) using inductive types. General `graph` model structure can be encoded using co-inductive types. However, as shown in [34] by C. Picard and R. Matthes, the encoding is quite complex as Coq enforces structural constraints when combining inductive and co-inductive types that forbid the use of the most natural encodings proposed by Poernomo et al. M. Giorgino et al. rely in [18] on a spanning tree of the graph combined with additional links to overcome that constraint using the Isabelle proof-assistant. This allows to develop a model transformation

[17] `http://alloy.mit.edu/alloy/`

relying on slightly adapted inductive proofs and then extract classical imperative implementations. Also, another implementation of the MDE is the HOL-OCL system [9] [10] that constitutes an environment for interactive modelling with UML and OCL and can be used for example to proof class invariants.

These embeddings are all shallow: they rely on sophisticated similar data structure to represent model elements and metamodels (e.g. Coq (co-)inductive data types for model elements and object and (co-)inductive types for metamodel elements). The work described in this paper is a deep embedding, each concept from models and metamodels was encoded in [44] using primitive constructs instead of relying on similar elements in Maude, Coq or Isabelle. The purpose of this contribution is not only to implement model transformation using correct-by-construction tools but also to give a kind of denotational semantics for the MDE concepts that should provide a deeper understanding and allow the formal validation of the various implemented technologies. Another formalisation in Coq of the MDE concepts by F.Barbier et al is accessible[18] [2], this representation is attached to the proof of the properties shown in [26] (instantiation relations and model transformations). Other work aiming to define a semantics for a modelling language by explicitly and denotationally defining the kind of systems the language describes and to focus on the variations and variability in the semantics [12] [28]. Compared to the last work, we are interested in a generic, complete and unique formalisation of the conformity to metamodels and we are focused mainly in the proof of the preservation of this conformity relation by the composition operators.

5.3 Compositional verification

The compositional verification, in other words to break up the verification of a system into the verification of its components, is a very old dream. Several work were conduced in this direction using the model checking technique.

For instance, Nguyen, T.H. proposed in [5] a compositional verification approach to check safety properties of component-based systems described in the BIP (Behavior - Interaction - Priority) language [3].

Also, another approach allowing to verify systems by composition from verified components was proposed in [47] where the temporal properties of a software component are specified, verified, and packaged with the component.

In this paper, regarding the previous cited methods, we adopted a generic composition technology that takes into account the EMOF metamodel properties making it usable with any language that can be described with a metamodel.

6 Conclusions

We have tackled in this paper the problem of model composition formalization and verification. In this purpose, starting from our formal framework for model

[18] http://web.univ-pau.fr/~barbier/Coq/

and metamodel formal specification Coq4MDE, we propose to rely on primitive composition operators that can then be used to build more sophisticated operators. We prove the correctness of the expected properties for these primitive ones introducing mandatory preconditions to reach compositional verification for the targeted properties. The proofs of property preservation for the high level operators combine the proofs of the primitive ones. Our proposal is validated in this contribution with the MOF model conformance property and the MOF package merge operator. All these notions are also currently reflected in the Coq proof-assistant, following the line of thought of our previous work around model and metamodel formalization. This embedding provides correct-by-construction pieces of executable code for the different model operations related to composition. As we target a general purpose MDE-oriented framework, our work applies to any model, modelling language, application and is not restricted to some more-or-less implicit language context.

This proposal is a preliminary mandatory step in the formalization of compositional formal verification technologies. We have tackled the formal composition of models independently of the properties satisfied by the model and the expected properties for the composite model. The next step in our work is to improve the notion of model compositional verification relying on several use cases from simple static constraints such as verification of OCL constraints satisfaction, to more dynamic properties such as deadlock freedom as proposed in the BIP framework [3].

In this last purpose, we need to model the behavioural part of each language. We propose to rely on the generic behaviours applicable to several meta-models sharing some features presented in [27]. This can be applied to families of unrelated meta-models. We plan to experiment the behavioural aspect by considering the merging of Statecharts Specifications [30]. In the long run, we plan to integrate the work of Garnacho et al. [17] that provide an embedding in Coq of timed transition systems in order to model the behavioral aspect of languages.

References

1. U. Aßmann. *Invasive software composition.* Springer-Verlag New York Inc, 2003.
2. F. Barbier, P. Castéran, E. Cariou, O. Le Goaer, et al. Adaptive software based on correct-by-construction metamodels. *Progressions and Innovations in Model-Driven Software Engineering*, pages 308–325, 2013.
3. A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in BIP. In *Software Engineering and Formal Methods, 2006. SEFM 2006. Fourth IEEE International Conference on*, pages 3–12. IEEE, 2006.
4. A. Baya and B. E. Asri. Composing specific domains for large scale systems. *Journal of Communication and Computer*, 10:844–856, 2013.
5. S. Bensalem, M. Bozga, T. Nguyen, and J. Sifakis. Compositional verification for component-based systems and application. *Software, IET*, 4(3):181–193, 2010.
6. P. Bernstein, A. Halevy, and R. Pottinger. A vision for management of complex models. *ACM Sigmod Record*, 29(4):55–63, 2000.
7. J. Bézivin. In search of a basic principle for model driven engineering. *Novatica Journal, Special Issue*, 5(2):21–24, 2004.

8. A. Boronat and J. Meseguer. An algebraic semantics for MOF. *Formal Aspects of Computing*, 22(3-4):269–296, 2010.
9. A. D. Brucker and B. Wolff. A proposal for a formal ocl semantics in isabelle/hol. In *Theorem Proving in Higher Order Logics*, pages 99–114. Springer, 2002.
10. A. D. Brucker and B. Wolff. Hol-ocl: a formal proof environment for uml/ocl. In *Fundamental Approaches to Software Engineering*, pages 97–100. Springer, 2008.
11. G. Brunet, M. Chechik, S. Easterbrook, S. Nejati, N. Niu, and M. Sabetzadeh. A manifesto for model merging. In *Proceedings of the 2006 international workshop on Global integrated model management*, pages 5–12. ACM, 2006.
12. M. V. Cengarle, H. Grönniger, B. Rumpe, and M. Schindler. System model semantics of class diagrams. *Technische Universitat Braunschweig*, 2008.
13. S. Clarke. Extending standard UML with model composition semantics. *Science of Computer Programming*, 44(1):71–100, 2002.
14. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285(2):187–243, 2002.
15. T. Coquand, G. Huet, et al. The calculus of constructions. 1986.
16. M. D. Del Fabro and P. Valduriez. Towards the efficient development of model transformations using model weaving and matching transformations. *Software and System Modeling*, 8(3):305–324, 2009.
17. M. Garnacho, J.-P. Bodeveix, and M. Filali-Amine. A mechanized semantic framework for real-time systems. In V. A. Braberman and L. Fribourg, editors, *FORMATS*, volume 8053 of *Lecture Notes in Computer Science*, pages 106–120. Springer, 2013.
18. M. Giorgino, M. Strecker, R. Matthes, and M. Pantel. Verification of the Schorr-Waite algorithm–from trees to graphs. In *Logic-Based Program Synthesis and Transformation*, pages 67–83. Springer, 2011.
19. M. K. Hamiaz, M. Pantel, B. Combemale, and X. Thirioux. Correct-by-construction model composition: Application to the invasive software composition method. In *FESCA*, pages 108–122, 2014.
20. J. Henriksson, F. Heidenreich, J. Johannes, S. Zschaler, and U. Aßmann. Extending grammars and metamodels for reuse: the Reuseware approach. *Software, IET*, 2(3):165–184, 2008.
21. J. Holt and S. Perry. *SysML for systems engineering*, volume 7. IET, 2008.
22. D. Jackson. Software abstractions-logic, language, and analysis, revised edition, 2012.
23. F. Jouault and J. Bézivin. Km3: A dsl for metamodel specification. In R. Gorrieri and H. Wehrheim, editors, *FMOODS*, volume 4037 of *Lecture Notes in Computer Science*, pages 171–185. Springer, 2006.
24. M. Kezadri. *Assistance à la validation et vérification de systèmes critiques: ontologies et intégration de composants*. PhD thesis, 2013.
25. M. Kezadri, B. Combemale, M. Pantel, X. Thirioux, et al. A proof assistant based formalization of components in MDE. In *8th International Symposium on Formal Aspects of Component Software (FACS 2011)*, 2011.
26. T. Kühne. Matters of (meta-) modeling. *Software & Systems Modeling*, 5(4):369–385, 2006.
27. J. Lara and E. Guerra. From types to type requirements: genericity for model-driven engineering. *Software and Systems Modeling*, 12(3):453–474, 2013.
28. S. Maoz, J. O. Ringert, and B. Rumpe. Semantically configurable consistency analysis for class and object diagrams. In *Model Driven Engineering Languages and Systems*, pages 153–167. Springer, 2011.

29. B. Morin, J. Klein, O. Barais, and J.-M. Jézéquel. A generic weaver for supporting product lines. In *Proceedings of the 13th international workshop on Early Aspects*, pages 11–18. ACM, 2008.
30. S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave. Matching and merging of statecharts specifications. In *Proceedings of the 29th international conference on Software Engineering*, pages 54–64. IEEE Computer Society, 2007.
31. Object Management Group, Inc. *Meta Object Facility (MOF) 2.0 Core Specification*, Jan. 2006. Final Adopted Specification.
32. Object Management Group, Inc. *Meta Object Facility (MOF) 2.4.2 Core Specification*, Jan. 2014.
33. O. OMG. Unified modeling language (omg uml)-infrastructure(v2.4.1). *Available on: http://www.omg.org/spec/UML/2.4.1*, 2.4.1, 2011.
34. C. Picard and R. Matthes. Coinductive graph representation : the problem of embedded lists. *Electronic Communications of the EASST, Special issue Graph Computation Models, GCM'10*, 2011.
35. I. Poernomo. The meta-object facility typed. In H. Haddad, editor, *SAC*, pages 1845–1849. ACM, 2006.
36. I. Poernomo. Proofs-as-model-transformations. In A. Vallecillo, J. Gray, and A. Pierantonio, editors, *ICMT*, volume 5063 of *Lecture Notes in Computer Science*, pages 214–228. Springer, 2008.
37. I. Poernomo and J. Terrell. Correct-by-construction model transformations from partially ordered specifications in Coq. In J. S. Dong and H. Zhu, editors, *ICFEM*, volume 6447 of *Lecture Notes in Computer Science*, pages 56–73. Springer, 2010.
38. J. R. Romero, J. E. Rivera, F. Durán, and A. Vallecillo. Formal and tool support for Model Driven Engineering with Maude. *Journal of Object Technology*, 6(9):187–207, 2007.
39. RTCA / EUROCAE. "Formal Methods Supplement to DO-178C [ED-12C]", DO-333/ED-218, 2011.
40. RTCA / EUROCAE. "Model-Based Development and Verification Supplement to DO-178C [ED-12C]", DO-331/ED-216, 2011.
41. RTCA / EUROCAE. "Software Considerations in Airborne Systems and Equipment Certification", DO-178C/ED-12C, 2011.
42. RTCA / EUROCAE. "DO-330/ED-215: Software Tool Qualification Considerations" - clarifying software tools and avionics tool qualification, 2012.
43. S. Sentilles, P. Štěpán, J. Carlson, and I. Crnković. Integration of extra-functional properties in component models. In *Component-Based Software Engineering*, pages 173–190. Springer, 2009.
44. X. Thirioux, B. Combemale, X. Crégut, and P.-L. Garoche. A Framework to Formalise the MDE Foundations. In R. Paige and J. Bézivin, editors, *International Workshop on Towers of Models (TOWERS)*, pages 14–30, Zurich, June 2007.
45. J. Troya and A. Vallecillo. Towards a rewriting logic semantics for ATL. In L. Tratt and M. Gogolla, editors, *ICMT*, volume 6142 of *Lecture Notes in Computer Science*, pages 230–244. Springer, 2010.
46. J. B. Warmer and A. G. Kleppe. *The object constraint language: getting your models ready for MDA*. Addison-Wesley Professional, 2003.
47. F. Xie and J. Browne. Verified systems by composition from verified components. *ACM SIGSOFT Software Engineering Notes*, 28(5):277–286, 2003.
48. A. Zito. *UML's Package Extension Mechanism: Taking a Closer Look at Package Merge*. Queen's University, 2006.