HAL
open science

# A Proof Assistant Based Formalization of components in MDE

Mounira Kezadri, Benoit Combemale, Marc Pantel, Xavier Thirioux

## To cite this version:

Mounira Kezadri, Benoit Combemale, Marc Pantel, Xavier Thirioux. A Proof Assistant Based Formalization of components in MDE. 8th International Symposium on Formal Aspects of Component Software (FACS 2011), University of Oslo, Norway, Sep 2011, Oslo, Norway. inria-00614593

**HAL Id: inria-00614593**

**https://inria.hal.science/inria-00614593v1**

Submitted on 30 Aug 2011

**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 proof assistant based formalization of components in MDE

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

[1] Université de Toulouse, IRIT - France
First name.Last name@enseeiht.fr
[2] Université de Rennes 1, IRISA, France,
First name.Last name@irisa.fr

**Abstract.** Model driven engineering (MDE) now plays a key role in the development of safety critical systems through the use of early validation and verification of models, and the automatic generation of software and hardware artifacts from the validated and verified models. In order to ease the integration of formal specification and verification technologies, various formalizations of the MDE technologies were proposed by different authors using term or graph rewriting, proof assistants, logical frameworks, etc.
The use of components is also mandatory to improve the efficiency of system development. Invasive Software Composition (ISC) has been proposed by Aßman in [1] to add a generic component structure to existing Domain Specific Modeling Languages in MDE. This approach is the basis of the ReuseWare toolset.
We present in this paper an extension of a formal embedding of some key aspects of MDE in set theory in order to formalize ISC and prove the correctness of the proposed approach with respect to the conformance relation with the base metamodel. The formal embedding we rely on was developed by some of the authors, presented in [23] and then implemented using the Calculus of Inductive Construction and the Coq proof-assistant. This work[3] is a first step in the formalization of composable verification technologies in order to ease its integration for DSML extended with component features using ISC.

## 1 Introduction

Model driven engineering now plays a key role in the development of safety critical systems through the use of model early validation and verification, and the automatic generation of software or hardware artefacts from the validated and verified models. This approach usually relies on many different Domain Specific Modeling Languages (DSML) either explicitly or through UML and its extensions that provides many different cooperating languages through diagrams (in fact, OMG is currently studying the possibility for the future next major

[3] This work was funded by the European Union and the french DGCIS through the ARTEMIS Joint Undertaking inside the CESAR project

version of UML to define it as a collection of cooperating DSML) and profiles. Each DSML is defined as a specific metamodel or as an extension through profiles of a part of a huge metamodel in UML.

The use of components is also mandatory to improve the efficiency of system development. Common DSML do not usually integrate components natively, either because it was not an initial requirement, or to avoid a too complex definition of the language. Invasive Software Composition (`ISC`) was proposed by Aßman [1] in order to add a generic component structure to any existing DSML. This approach is the basis of ReuseWare[4] that provides ISC based tools inside the Eclipse Modeling Framework[5]. It allows to define the composition concern relying on elements in the metamodel and then to extract components from existing models with defined composition interface (called fragment boxes), and to compose fragments to produce new fragments or models. All the provided tools are generic and parametrized by the composition concern. The framework allows to adapt and extend an existing language by adding composition facilities at some points called `Hook`. This extension relies on a metamodel level transformation applied on the language definition based on the specification of the composition concern. The `Hook` are the variation points introduced in the models whose value can change and thus allows to build components. The main advantage of the ISC technology is that it is generic and can be applied to any language defined by a metamodel. This framework ensures that the result of the composition of fragments extracted from models conforming to a given metamodel is also conforming to the same metamodel. This common conformance is the kind of standard structural properties available in all the MDE frameworks that is verified in this paper. The long term purpose of our work is also to handle behavioral properties and thus tackle the formalization of all kind of compositional verification technologies.

In order to ease the integration of formal specification and verification technologies, some of the authors proposed in [23] a formal embedding of some key aspects of Model Driven Engineering in Set Theory. This embedding was then implemented using the Calculus of Inductive Construction and the Coq[6] proof-assistant. This first version focused on the notions of models, metamodels, conformance and promotion. It was later extended to express constraints on metamodels using the Object Constraint Language (OCL). The purpose of this framework called Coq4MDE is to provide sound mathematical foundations for the study and the validation of MDE technologies. The choice of constructive logic and 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 model driven engineering (for example, in the Eclipse Modeling Project).

This paper contributions are the specification of the composition operators provided by the `ISC` method [1] using an extension of Coq4MDE and the proof

[4] `http://www.reuseware.org`
[5] `http://www.eclipse.org/modeling/emf`
[6] `http://coq.inria.fr`

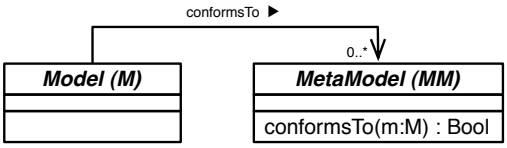


**Fig. 1.** Model & MetaModel Definition using the UML Class Diagram Notation

of the well-foundedness and termination of these operators. This specification allows to express the models expected properties and the verification technologies for composite models and then provide support for compositional verification. This first contribution focuses on the metamodel structural conformance relation. It relies on the `Model` and `MetaModel` concepts from Coq4MDE that is extended to represent fragments as proposed by ISC. The various concepts provided by ReuseWare are formalized leading to the proof that composition preserves metamodel conformity.

First, Section 2 introduces the notions of $Model$ and $MetaModel$ from Coq4MDE. Then, the ReuseWare approach for extending DSML with components is presented in Section 3. The Coq4MDE framework is then extended to support the definition of component interface and the composition operators in Section 4. After that, the validation of a composition function is presented in Section 5. Also, a background of related work is given in Section 6. Finally, conclusion and perspectives are presented in Section 7.

## 2 Model and MetaModel

This section gives the main insight of our MDE framework Coq4MDE, derived from [23]. We first define the notions of *model* and *metamodel*. Then, we describe conformity using the *conformsTo* predicate.

Our approach separates the type level from the instance level, and describes them with different structures hence different types. A *Model* ($M$) is the instance level and a *MetaModel* ($MM$) is a modeling language used to define models (Figure 1). A $MM$ also specifies the semantic properties of its models. For instance, in UML, a multiplicity is defined on relations to specify the allowed number of objects that have to be linked. Moreover, OCL is used to define more complex structural constraints which may not have any specific graphical notation.

Into our framework, the concept of $MetaModel$ is not a specialization of $Model$. They are formally defined in the following way. Let us consider two sets: `Classes`, respectively `References`, represents the set of all possible class, respectively reference, labels. We also consider instances of such classes, the set `Objects` of object labels. `References` includes a specific *inh* label used to specify the inheritance relation. In the following text, we will withdraw the word label and directly talk about classes, references and objects.

**Definition 1 (Model).** *Let $\mathscr{C} \subseteq \texttt{Classes}$ be a set of classes.*
*Let $\mathscr{R} \subseteq \{\langle c_1, r, c_2\rangle \mid c_1, c_2 \in \mathscr{C}, r \in \texttt{References}\}$ be the set of references among classes such that $\forall c_1 \in \mathscr{C}, \forall r \in \texttt{References}$, $card\{c_2 \mid \langle c_1, r, c_2\rangle \in \mathscr{R}\} \leq 1$.*

*A model over $\mathscr{C}$ and $\mathscr{R}$, written $\langle MV, ME\rangle \in \texttt{Model}(\mathscr{C}, \mathscr{R})$ is a multigraph built over a finite set $MV$ of typed object nodes and a finite set $ME$ of reference edges such that:*

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

Note that, in case of inheritance, the same object label will be used several time in the same model graph, associated to different classes to build different nodes. This label reuse is related to inheritance polymorphism a key aspect of most OO languages. Inheritance is represented with a special reference called *inh* [7] (usually defined in the metamodeling languages such as MOF [17]).

Accordingly, we first define an auxiliary predicate stating that an object $o$ of type $c_1$ has a downcast duplicate of type $c_2$.

$$\begin{aligned} &hasSub(o \in \texttt{Objects}, c_1, c_2 \in \texttt{Classes}, \langle MV, ME\rangle) \triangleq \\ &\quad c_1 = c_2 \vee \exists c_3 \in \texttt{Classes}, \langle\langle o, c_2\rangle, inh, \langle o, c_3\rangle\rangle \in ME \\ &\quad \wedge hasSub(o, c_1, c_3, \langle MV, ME\rangle) \end{aligned}$$

Then, we define the notion of standard inheritance. The first part of the conjunction states that the inheritance relation only conveys duplicate objects. The second part states that every set of duplicates has a common base element (a common inherited class).

$$\begin{aligned} &standardInheritance(\langle MV, ME\rangle) \triangleq \\ &\quad \forall\langle\langle o_1, c_1\rangle, inh, \langle o_2, c_2\rangle\rangle \in ME, o_1 = o_2 \\ &\quad \wedge\forall\langle o_1, c_1\rangle, \langle o_2, c_2\rangle \in MV, o_1 = o_2 \Rightarrow \exists c \in \texttt{Classes}, \\ &\quad\quad hasSub(o_1, c_1, c, \langle MV, ME\rangle) \\ &\quad\quad \wedge hasSub(o_2, c_2, c, \langle MV, ME\rangle) \end{aligned}$$

Finally, the following property states that $c_2$ is a direct subclass of $c_1$.

$$\begin{aligned} &subClass(c_1, c_2 \in \texttt{Classes}, \langle MV, ME\rangle) \triangleq \\ &\quad \forall\langle o, c\rangle \in MV, c = c_2 \Rightarrow \langle\langle o, c_2\rangle, inh, \langle o, c_1\rangle\rangle \in ME \end{aligned}$$

Consequently, *Abstract Classes*, that are specified in the metamodel using the *isAbstract* attribute, serve as parent classes and child classes are derived from them. They are not themselves suitable for instantiation. Abstract classes are often used to represent abstract concepts or entities. Features of an abstract class are then shared by a group of sibling sub-classes which may add new properties.

Therefore, a model does not conform to a metamodel if it contains objects that are instances of abstract classes without having instances of concrete derived classes as duplicates.

$$\begin{aligned} &isAsbstract(c_1 \in \texttt{Classes}, \langle MV, ME\rangle) \triangleq \\ &\quad \forall\langle o, c\rangle \in MV, c = c_1 \Rightarrow \exists c_2 \in \texttt{Classes}, \langle\langle o, c_2\rangle, inh, \langle o, c_1\rangle\rangle \in ME \end{aligned}$$

[7] *inh* must not be used in a model or metamodel as a simple reference

**Definition 2 (MetaModel).** *A MetaModel is a multigraph representing classes and references as well as semantic 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 class nodes and a finite set* $MME$ *of edges tagged with references, and of a predicate over models representing the semantic properties.*

*A metamodel as a pair* $\langle (MMV, MME), conformsTo \rangle \in \texttt{MetaModel}$ *such that:*

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

*such that* $\forall c_1 \in MMV, \forall r \in \texttt{References}, card\{c_2 \mid \langle c_1, r, c_2 \rangle \in MME\} \leq 1$

Given one model $M$ and one metamodel $MM$, we can check conformance. The $conformsTo$ predicate embedded in $MM$ achieves this goal. It identifies the set of valid models with respect to a metamodel.

In our framework, the conformance checks on the model $M$ that:

1. every object $o$ in $M$ is the instance of a class $C$ in $MM$.
2. every link between two objects is such that there exists, in $MM$, a reference between the two classes typing the two elements. In the following we will say that these links are instances of the reference between classes in $MM$.
3. finally, every semantic property defined in $MM$ is satisfied in $M$. For instance, the multiplicity defined on references between concepts denotes a range of possible links between objects of these classes (i.e. concepts). Moreover, structural properties expressed on the metamodel as OCL constraints and behavioural properties will be taken into account in future work as $conformsTo$ predicates.

This notion of conformity can be found in the framework depicted in Figure 1 by a dependency between a $M$ and a $MM$ it conforms to. In fact, the semantic properties associated to the metamodel are encoded into the $conformsTo$ predicate. These semantic properties are not to be given a syntax. Instead, in order to express our properties, we assume an underlying logic that should encompass OCL in terms of expressive power.

In the rest of this paper, we extend the previous MDE framework to formalize compositional technologies. Our final target outside the scope of this paper is to formalize compositional verification activities. Coq4MDE is extended to support the introduction of components in DSML defined by their metamodels. This extension allows to express fragment boxes (models with defined interface) composition based on concepts from the `ISC` method.

In the scope of this paper, we take into account a simplified version of the $conformsTo$ predicate (cf. Section 5) called $instanceOf$ which is restricted to 1 and 2. We demonstrate that the verification of this $instanceOf$ property is compositional relying on the `ISC` operators (the property of components is preserved in case of composition using the `ISC` basic operators).

## 3 ISC and ReuseWare approach

ISC [1] is a generic technology for extending a DSML with model composition facilities. Its first version was defined to compose Java programs and was implemented in the COMPOST system[8]. A universal extension called U-ISC was proposed in [12], this technique deals with textual components that can be described using context-free grammars and then the fragments are represented as trees. The method as presented considers tree merging for the composition. Recently, in order to deal with graphical languages the method was extended to support typed graphs in [14], this method was implemented in the ReuseWare framework. This last implementation is consistent with the description of models as graphs in our Coq4MDE framework.

ISC introduces the fragment box structure to group model or source code fragments. The fragment box defines its composition interface and then provides tools and concepts allowing the composition. The composition interface for a fragment box consists of a set of addressable points. Two types of addressable points are defined, the variation points which are elements inside the fragment box that can be used as a receptor for other elements and reference points which are used to address some parts inside a fragment box so they can be used in composition. We formalize thereafter one type of correspondence (variation/reference) points which is the pair (hook/prototype). As described in [11] a hook is a variation point that constitutes a place-holder to contain a fragment referenced by a prototype reference point.

We propose in the following section to extend the Coq4MDE framework to support ISC concepts and then to define a sound basis to ensure the correctness by construction for this composition style. This enables to describe and to verify structural properties. We plan in future work to extend the formalization to support other kind of properties and especially behavioural properties.

## 4 Formalizing Model Component Extraction and Composition

### 4.1 Extended MetaModel with Model Component

We must be able to extend any metamodel to support the definition of fragment boxes. This extension adds the definition of a fragment interface constituted from a set of addressable points. We note the extended metamodel for some metamodel $MM$ as $MM^{Ext}$. We note $ROV$ the abstract class representing the addressable points, the $Hook$ variation point and the $Prototype$ reference points are subclasses of $ROV$. In $MM^{Ext}$, every node in the graph representing $MM$ can be referenced by an addressable point. For this purpose, an abstract class called $AbsC$ is added as a super class for all the classes of $MM$. This class is linked by the reference $bind$ with $ROV$. The three classes $ROV$, $Hook$ and

[8] http://www.the-compost-system.org

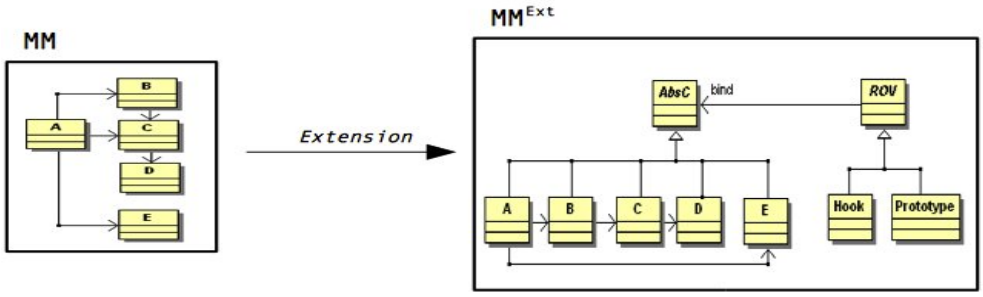


**Fig. 2.** MetaModel extension

*Prototype* are also automatically imported to the metamodel with appropriate inheritance relations between them [9].

The following definition represents the extension function implemented in Coq as a graph transformation which is not in the scope of this paper.

**Definition 3.** *Let* $MM = \langle\langle MMV, MME\rangle, conformsTo\rangle$ *be a metamodel.*
*Let* $ROV, Hook, Prototype, AbsC \in Classes, bind \in References.$
$MM^{Ext}$ *is defined as* $\langle\langle MMV^{Ext}, MME^{Ext}\rangle, conformsTo^{Ext}\rangle$ *such that:*

$$
\begin{aligned}
&MMV^{Ext} = MMV \cup \{ROV, Hook, Prototype, AbsC\}\\
&MME^{Ext} = MME \cup \{\langle ROV, bind, AbsC\rangle\}\\
&conformsTo^{Ext}(\langle MV, ME\rangle) \triangleq conformsTo(\langle MV, ME\rangle)\\
&\wedge\ isAbstract(ROV)\\
&\wedge\ subClass(Hook, ROV)\\
&\wedge\ subClass(Prototype, ROV)\\
&\wedge\ isAbstract(AbsC)\\
&\wedge\ \forall c \in MMV, subClass(c, AbsC)
\end{aligned}
$$

The figure 2 shows the example of the extension of the MetaModel $MM$.

### 4.2 Component interface extraction

The goal of the function `FragmentExtraction` is to construct a fragment box from a model by defining its composition interface. This function takes as parameters: a model, the object referenced in that model and the kind of the addressable point associated to this object.

[9] The metamodel extension used in [14] is defined at the third modeling level (metametamodel level) which may use the promotion notion to be defined in the Coq4MDE framework. The extension defined thereafter uses only the second modeling level (metamodel level) which seems to be sufficient.

$FragmentExtraction : Model \times \texttt{Objects} \times \texttt{Classes} \rightarrow Model$ is defined as[10]:

$$
\begin{array}{l}
FragmentExtraction(\langle MV, ME\rangle, o, HP) = \ \langle MV^{Ext}, ME^{Ext}\rangle \\
where\ HP \in \{Hook, Prototype\}\ and\ \exists c \in \texttt{Classes}, \langle o, c\rangle \in MV \\
such\ that: \\
MV^{Ext} = MV \cup \{\langle h, HP\rangle, \langle h, ROV\rangle, \langle o, AbsC\rangle\} \\
ME^{Ext} = ME \cup \{\langle\langle o, c\rangle, inh, \langle o, AbsC\rangle\rangle, \\
\langle\langle h, ROV\rangle, bind, \langle o, AbsC\rangle\rangle, \\
\langle\langle h, HP\rangle, inh, \langle h, ROV\rangle\rangle\}
\end{array}
$$

`ElimInterface` eliminates the fragment box interface (all variation and reference points) of a fragment box, it is the inverse function of `FragmentExtraction` in case of only one addressable point in the fragment box. This is implemented in [14] using the *remove* operator which is automatically applied after composition execution to make the component understandable by tools where addressable points semantics is not defined.
$ElimInterface : Model \rightarrow Model$, such as:

$$
\begin{array}{l}
ElimInterface\ \langle MV^{Ext}, ME^{Ext}\rangle = \ \langle MV, ME\rangle \\
such\ that: \\
MV = \{\langle o, c\rangle \in MV^{Ext} | c \notin \{Hook, Prototype, VOR, AbsC\}\} \\
ME = \{\langle\langle o, c\rangle, r, \langle o', c'\rangle\rangle \in ME^{Ext} | c, c' \notin \{Hook, Prototype, ROV, AbsC\}\}
\end{array}
$$

The definition of these two functions requires some proofs on multigraphs. First, the proof that the extension of the multigraph representing the model is also a multigraph [11], this is done by proving that adding vertexes to a multigraph generates a multigraph and also adding edges in some conditions to a multigraph is also a multigraph. Second, the proof that deleting some elements from a multigraph representing the fragment box is also a multigraph [12], this is done using a filter function defined on multigraphs. So, Coq4MDE can now support the definition of components with composition interface in any DSML. We describe in the following section the formalisation of `ISC` basics composition operators in Coq4MDE.

### 4.3 Components Composition

In this section, we present the implementation in our framework of the two basic operators of `ISC` (*bind* and *extend*) presented in [1] [14] . The difference between these operators is that "the *bind* applied to the hook replaces the hook (i.e., it removes the hook from its containing fragment) while *extend* applied on a hook does not modify the hook itself but uses it as a position for extension (i.e., the hook remains in its containing fragment) ".

[10] Another version can be implemented by specifying a set of pairs $(o, HP)$ to add several points at the same time.
[11] `http://www.irit.fr/~Mounira.Kezadri/FISC/MMext.html`
[12] `http://www.irit.fr/~Mounira.Kezadri/FISC/IntElim.html#elimInterface`

**Bind** The bind operator replaces an object $o1$ referenced by a hook variation point by an object $o2$ referenced by a prototype reference point. The links to (resp. from) the object $o1$ are replaced with links to (resp. from) the object $o2$. The composed model is obtained by substituting the object $o1$ by $o2$ in both objects and links sets. $bind : Model \times Model \times (Objects \times Classes) \times (Objects \times Classes) \rightarrow Model$ is defined as:

$$
\begin{array}{l}
bind(\langle MV1, ME1\rangle, \langle MV2, ME2\rangle, \langle b, B\rangle, \langle b', B'\rangle) = \langle MV3, ME3\rangle \\
where\ \langle b, B\rangle \in MV1\ and\ \langle b', B'\rangle \in MV2,\ we\ have: \\
\exists h, p \in Objects, \langle\langle h, Hook\rangle, inh, \langle h, ROV\rangle\rangle \in ME1 \\
\wedge\langle\langle h, ROV\rangle, bind, \langle b, AbsC\rangle\rangle \in ME1 \\
\wedge\langle\langle b, B\rangle, inh, \langle b, AbsC\rangle\rangle \in ME1 \\
\wedge\langle\langle p, Prototype\rangle, inh, \langle p, ROV\rangle\rangle \in ME2 \\
\wedge\langle\langle p, ROV\rangle, bind, \langle b', AbsC\rangle\rangle \in ME2 \\
\wedge\langle\langle b', B'\rangle, inh, \langle b', AbsC\rangle\rangle \in ME2 \\
and\ finally: \\
MV3 = substV(\langle b, B\rangle, \langle b', B'\rangle, MV1) \\
ME3 = substE(\langle b, B\rangle, \langle b', B'\rangle, ME1)
\end{array}
$$

such that $substV(\langle b, B\rangle, \langle b', B'\rangle, MV)$ (resp. $substE(\langle b, B\rangle, \langle b', B'\rangle, ME)$) is the function that replaces $\langle b, B\rangle$ by $\langle b', B'\rangle$ in every element in `MV` (resp. relation in `ME`). The condition of the composition is: $B = B'$.

The construction of this function in Coq requires the proof that substituting an object by another in some multigraph is also a multigraph [13]. The proof is done by induction, it is automatic for the empty graph. In case of a graph built from adding an edge (a reference) to the graph, one reference is presented as $\langle src, dst, a\rangle$, suppose that the substitution replaces $o1$ by $o2$, we must consider all cases of equality between $src$, $dst$, $o1$ and $o2$. Last, in case of a graph built by adding a vertex to a graph which considers also cases of equality between the added vertex, $o1$ and $o2$. The current implementation can be largely improved by the definition of some graph operations like the map function, which is currently partially done and will be presented in future work. A recursive call for the previous function using a list of correspondence (Variation/Reference) points allows to replaces several objects at the same time.

**Extend** This operator allows to extend a model $\langle MV1, ME1\rangle$ (the extension point is an object $o1$ addressed as a hook variation point inside the model) by a model $\langle MV2, ME2\rangle$ at an object $o2$ addressed as a prototype reference point.

This function is parametrized by a metamodel (to insure the type safety) and a name for the added link between $o1$ and $o2$. The composed model consists of a multigraph built over the union of all objects of $\langle MV1, ME1\rangle$ and $\langle MV2, ME2\rangle$, all links of the two models in addition to a link between the objects $o1$ and $o2$.

$extend : Model \times Model \times (Objects \times Classes) \times (Objects \times Classes) \times MetaModel \times References \rightarrow Model$ is defined as:

[13] `http://www.irit.fr/~Mounira.Kezadri/FISC/CompBind.html#GraphSubst`

$$
\begin{array}{l}
extend(\langle MV1, ME1\rangle, \langle MV2, ME2\rangle, \langle b, B\rangle, \langle b', B'\rangle, \\
(\langle MMV, MME\rangle, conformsTo), LinkName) = \langle MV3, ME3\rangle \\
where\ \exists\ \langle b, B\rangle \in MV1\ and\ \langle b', B'\rangle \in MV2,\ we\ have: \\
extensible(\langle MV1, ME1\rangle, \langle MV2, ME2\rangle, \langle b, B\rangle, \langle b', B'\rangle, \\
(\langle MMV, MME\rangle, conformsTo), LinkName)\ such\ that: \\
MV3 = MV1 \cup MV2 \\
ME3 = ME1 \cup ME2 \cup \{\langle\langle b, B\rangle, LinkName, \langle b', B'\rangle\rangle\}
\end{array}
$$

The predicate *extensible* checks that a model $\langle MV1, ME1\rangle$ whose interface is $\langle b, B\rangle$ regarding some metamodel can be extended by another model $\langle MV2, ME2\rangle$ whose interface is $\langle b', B'\rangle$.

$$
\begin{array}{l}
extensible(\langle MV1, ME1\rangle, \langle MV2, ME2\rangle, \langle b, B\rangle, \langle b', B'\rangle, \\
(\langle MMV, MME\rangle, conformsTo), LinkName) \triangleq \\
isExtendedH(\langle MV1, ME1\rangle, \langle b, B\rangle) \\
\wedge isExtendedP(\langle MV1, ME1\rangle), \langle b', B'\rangle) \\
\wedge (B, LinkName, B')\ \in\ MME
\end{array}
$$

The predicate *isExtendedH* verifies that $\langle b, B\rangle$ is a hook in $\langle MV1, ME1\rangle$.

$$
\begin{array}{l}
isExtendedH\langle MV1, ME1\rangle\langle b, B\rangle \triangleq \\
\exists h \in Objects, \langle\langle h, Hook\rangle, inh, \langle h, ROV\rangle\rangle \in ME1 \\
\wedge\langle\langle h, ROV\rangle, bind, \langle b, AbsC\rangle\rangle \in ME1 \\
\wedge\langle\langle b, B\rangle, inh, \langle b, AbsC\rangle\rangle \in ME1
\end{array}
$$

The predicate *isExtendedP* verifies that $\langle b, B\rangle$ is a prototype in the model.

$$
\begin{array}{l}
isExtendedP\langle MV2, ME2\rangle\langle b, B\rangle \triangleq \\
\exists p, \langle\langle p, Prototype\rangle, inh, \langle p, ROV\rangle\rangle \in ME2 \\
\wedge\langle\langle p, ROV\rangle, bind, \langle b, AbsC\rangle\rangle \in ME2 \\
\wedge\langle\langle b, B\rangle, inh, \langle b, AbsC\rangle\rangle \in ME2
\end{array}
$$

The construction of this function in Coq requires the proof that the multigraph built by extending another multigraph as described in the function *extend* is also a multigraph [14].

Here we defined only one type of correspondence variation and reference point (hook/prototype), the method as presented in [14] considers also another type of correspondence (slot/anchor). The second type requires to consider the containment property of an edge. The difference as explained in [14] is that contrarily to hook and prototype the slot variation point and the anchor reference point keeps their containments in case of composition. The first type of correspondence allows to express quite complicated composition functions like described in the following example and is consistent with the current models graph representation. The second type of correspondence can be considered in future work. The operators like described here are applied to the two models, a generalization to

[14] `http://www.irit.fr/~Mounira.Kezadri/FISC/CompBind.html#compositionExtend`

an application on several models at the same time is allowed in ReuseWare and can be implemented in our framework as an iterative application of the operators by composing the models one by one or by defining more general operators that can be applied on several models.

### 4.4 Detailed example

We describe in this section the use of the previously defined basic operators to elaborate a model composition. $M_1$ is a state machine modeling a door with a lock. The door provides the operations: open, close, pass, lock and unlock. We would like to add the possibility of simple and double locking the door, these two states are described in the model $M_2$. $M_1$ and $M_2$ are described in Fig. 3.

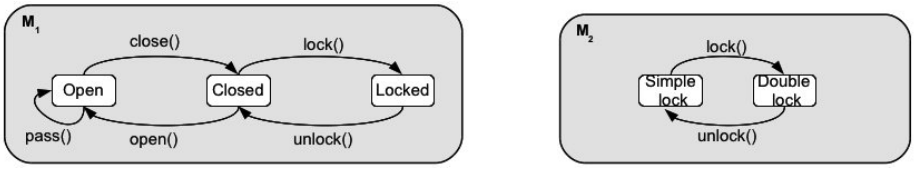


**Fig. 3.** $M_1$ and $M_2$ models

The first step is to define the interface for each model. This is done with the *FragmentExtraction* function, the function applied to the model $M_1$ defines *Locked* as a hook and applied to $M_2$ defines *Simple lock* as a prototype like described in Fig. 4.

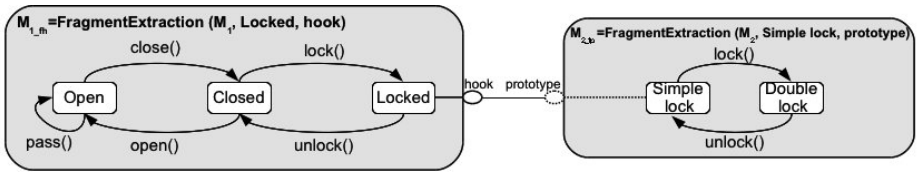


**Fig. 4.** Variation and reference point for the models $M_1$ and $M_2$

The application of the function *bind* on the two fragments as described in Fig. 4 followed by the elimination of the interface produces the model $M_{bind}$ shown in Fig. 5.

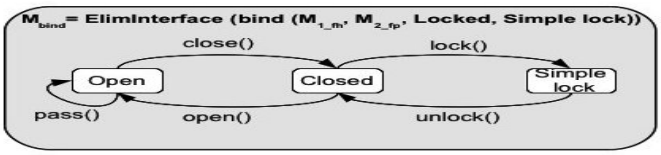


**Fig. 5.** Model after execution of the *bind* function

Then, *Simple lock* is defined in $M_{bind}$ as a prototype reference point and *Double lock* is defined in $M_{2_fp_elim}$ as a hook variation point as shown in Fig. 6.

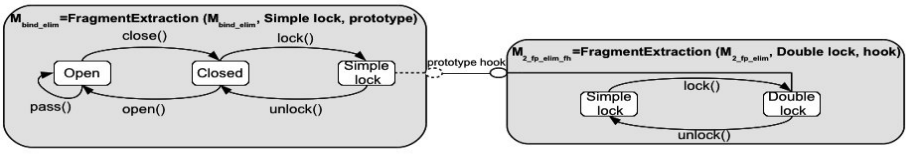


**Fig. 6.** Fragment boxes extraction

The execution of the function *extend* on the two models in Fig. 6 after the interface elimination generates the model presented in Fig. 7. The model is the state machine for a door with a double lock option.

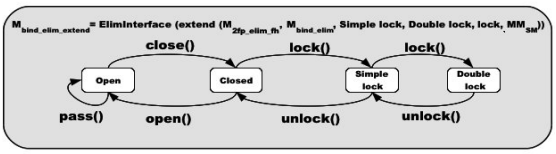


**Fig. 7.** Model after execution of the *extend* and $ElimInterface$ functions

The original contribution of this paper is not the definition of composition operators which is taken from `ISC` but their implementation in the Coq proof assistant, their integration in the Coq4MDE framework and the proof that the verification of the $instanceOf$ property is compositional with respect to these operators.

## 5 Composition Validation

The `bind` and `extend` operators are defined in order to enforce the well typedness properties. These two operators like all the concepts presented in this paper are encoded in the Coq proof assistant. The aim of this formalization is to check some properties on the composite models and then provide the basis for the specification and proof of correctness of compositional verification technologies. The first property considered is the well typedness property. This property is related to the conformance defined in Section2. It checks that every object in $M$ is the instance of a class in $MM$ and every link in $M$ is an instance of a relation in $MM$. To prove that this verification is compositional, we need to prove that the composition of two models instances of the same metamodel is also an instance of the same metamodel.

We define the first validity criteria for any composition function. This criteria is defined as a higher order predicate that checks the well typedness for some function. The function `InstanceOf` is used in that purpose, it checks that

all objects and links of a $Model$ are instances of classes and references in a metamodel.

$$
\begin{aligned}
&InstanceOf(\langle\langle MV, ME\rangle, \langle\langle MMV, MME, conformsTo\rangle\rangle\rangle) \triangleq \\
&\forall\langle o, c\rangle \in MV, c \in MMV \wedge \\
&\forall\langle\langle o, c\rangle, r, \langle o', c'\rangle\rangle \in ME \wedge \langle c, r, c'\rangle \in MME
\end{aligned}
$$

Then, the predicate $\texttt{ValidCompositionFunction}_{MM}$ reflects this criteria. It verifies that using two components instance of $MM$, the component resulting from the application of a composition function `f` is also instance of $MM$.

$$
\begin{aligned}
&ValidCompositionFunction(MM \in MetaModel, f) \triangleq \\
&\forall\ M1\ M2 \in Model, \\
&InstanceOf\ (M1, MM)\ \wedge InstanceOf\ (M2, MM) \\
&\rightarrow InstanceOf\ ((f\ M1\ M2), MM)
\end{aligned}
$$

We use this predicate to verify the type safety for the composition operator *bind* described in Section 4.3. This is described in the theorem `ValidBind`.

$$
\begin{aligned}
&\texttt{Theorem}\ ValidBind : \forall\ MM \in MetaModel, \\
&ValidCompositionFunction(MM, bind)
\end{aligned}
$$

The Coq proof is done for this theorem. It uses intermediate lemmas that prove the preservation of the well typedness by the elementary operations implied in the composition. Among these lemmas, `conformsAddO` ensures that the result of adding an object instance of a class in the metamodel to a component instance of this metamodel is a component instance of the same metamodel.

$$
\begin{aligned}
&\texttt{Theorem}\ conformsAddO : \\
&\forall\langle MV, ME\rangle \in Model, \langle(MMV, MME), conformsTo\rangle \in MetaModel. \\
&\forall o \in \texttt{Objects}, c \in \texttt{Classes}. \\
&InstanceOf(\langle MV, ME\rangle, \langle(MMV, MME), conformsTo\rangle) \wedge c \in MMV \\
&\rightarrow InstanceOf(\langle MV \cup \{\langle o, c\rangle\}, ME\rangle, \langle(MMV, MME), conformsTo\rangle)
\end{aligned}
$$

Another Coq proof was done to demonstrate the type safety for the composition operator *extend* described also in Section 4.3. This is encoded in the theorem `ValidExtend`.

$$
\begin{aligned}
&\texttt{Theorem}\ ValidExtend : \forall\ MM \in MetaModel, \\
&ValidCompositionFunction(MM, extend)
\end{aligned}
$$

Also, similar correction properties should hold for the fragment extraction function and the elimination function.

$$
\begin{aligned}
&\texttt{Theorem}\ ValidFragmentExtraction : \\
&\forall\langle MV, ME\rangle \in Model, \langle(MMV, MME), conformsTo\rangle \in MetaModel. \\
&\forall o \in \texttt{Objects}, HP \in \{Hook, Prototype\}. \\
&InstanceOf(\langle MV, ME\rangle, \langle(MMV, MME), conformsTo\rangle) \\
&\rightarrow InstanceOf(FragmentExtraction(\langle MV, ME\rangle, o, HP), \\
&\langle(MMV^{Ext}, MME^{Ext}), conformsTo^{Ext}\rangle)
\end{aligned}
$$

$\texttt{Theorem}\ ValidInterfaceElimination:$
$\forall \langle MV, ME\rangle \in Model, \langle (MMV, MME), conformsTo\rangle \in MetaModel.$
$InstanceOf(\langle MV, ME\rangle, \langle (MMV^{Ext}, MME^{Ext}), conformsTo^{Ext}\rangle)$
$\rightarrow InstanceOf(InterfaceElimination(\langle MV, ME\rangle),$
$\langle (MMV, MME), conformsTo\rangle)$

So, starting from the Coq4MDE framework and from the `ISC` composition method, we defined a framework for model composition. The definitions of model and metamodel were extended to support the definition of model composition interface, the constituted fragment box is also a model conforms to an extended metamodel. The basic composition operators was described like all elements in this paper using the Coq proof assistant. The source code is about 6400 lines, it is accessible at `http://www.irit.fr/~Mounira.Kezadri/FISC/index.html`. The formalization in Coq ensures the termination[15] of the composition operators, elaborates a compositional verification property and also will enable to describe and prove more richer properties in future work.

## 6 Related work

### 6.1 Composition approaches

Models are aspects of the system that must be composed to build the final system, similarly to aspects in AOP [15]. Tools and approaches have been proposed aiming to automate the composition task. This problem concerns a wide variety of modeling domains and includes several techniques. We are looking for an approach that supports component extraction from models and model composition from components. The `ISC` approach supports these two characteristics. It enables to extend arbitrary language to provide reuse with the concepts of fragment box. In this method components can be invasively composed, this can be done by adapting or extending the component at some variation point (fragments or positions, which are subject to change) by transformation. Several composition methods were collected in[13]. most of these methods are interested in implementing the merge operator by using some mappings between the models like Rational Software Architect [16] , Bernstein et al. data model [5], Atlas Model Weaver [17] [9], Epsilon [18], Theme/UML [7] and EMF Facet[19]. Merge operators as presented in these works can be implemented in our framework and constitutes one of the directions for future work.

[15] We can't write any function in Coq if the proof of termination is not given or deduced by Coq
[16] http://www-306.ibm.com/software/awdtools/architect/swarchitect/
[17] http://www.eclipse.org/gmt/amw/
[18] http://www.eclipse.org/gmt/epsilon/
[19] www.eclipse.org/proposals/emf-facet/

### 6.2 Formalization of model driven engineering

MoMENT (MOdel manageMENT) [6] is a model management framework based on experiments in formal model transformation and data migration, it provides a set of generic operators to manipulate models. MoMENT relies on algebraic formalisms using the Maude language [8]. In this framework, the metamodels are represented as algebraic specifications and the operators are defined independently of the metamodel. To be used, the operators must be specified in a module called signature that specify the constructs of the metamodel. The approach was implemented in a tool [20] that gives also an automatic translation from an EMF metamodel to a signature model.

A. Vallecillo et al. have designed and implemented previously a different embedding of metamodels, models ([22]) and model transformations ([24]) 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.

I. Poernomo has proposed an encoding of metamodels and models using type theory ([19]) in order to allow correct by construction development of model transformation using proof assistant like Coq ([20]). Some simple experiments have been conducted using Coq mainly on tree-shaped models ([21]) using inductive types. General graph model structure can be encoded using co-inductive types. However, as shown in [18] 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 [10] 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 relying on slightly adapted inductive proofs and then extract classical imperative implementations. 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 are encoded using elementary constructs instead of relying on similar elements in MAUDE, Coq or Isabelle. The purpose of this contribution is not to implement model transformation using correct-by-construction tools but to give a kind of denotational semantics for model driven engineering concepts that should provide a deeper understanding and allow the formal validation of the various implemented technologies.

### 6.3 Formalization of models composition

A formalisation of `ISC` in Frame Logic (or F-Logic) [16] was proposed in [2]. F-Logic provides structural aspects of object oriented and frame-based languages (object identity, complex objects, inheritance, polymorphic types, query

[20] http://moment.dsic.upv.es/

methods, encapsulation and others). The description in F-Logic allows reasoning on the composition architecture and provides many additional checking: cyclic check, reachability and constraint check. In this work we define the mathematical formalisation of the concepts of the ISC method aiming to describe it in a proof assistant. The advantage of this formalization in addition to those of the previous cited work (it can be added to any model and is independent of specific component description languages and the checked properties), are proof of termination of composition functions and the possibility of extracting the validated executable code from the definitions after some modifications on functions that are written now for validation purpose.

### 6.4 Compositional verification

In order to develop safety critical systems, methods are now needed that allows not only the reuse of components but also of their properties for inferring the global properties of the composite system from properties of his constituent components. Nguyen, T.H. proposes in [4] a compositional verification approach to check safety properties of component-based systems. The systems must be described in the BIP (Behavior - Interaction - Priority) language [3]. Another approach allowing to verify systems by composition from verified components was proposed in [25], this approach reduces the complexity of verifying component-based systems by utilizing their compositional structures. In this approach, temporal properties of a software component are specified, verified, and packaged with the component. The selection of a component for reuse considers also its temporal properties. The Ptolemy[21] project proposes a compositional theory for concurrent, real-time, embedded systems. It uses well defined models of computation and defines an unified mathematical framework to relate heterogeneous models of computation. In this paper, regarding the previous cited methods, we adopted a generic composition technology where the interactions and temporal properties are not yet integrated. This is planned for future work.

## 7 Conclusion

Starting from Coq4MDE our formal framework for model and metamodel definition, we have tackled the problem of model composition. Taking inspiration from the ISC generic method for model composition and also from the ReuseWare toolbox, we proposed first a metamodel extension, and associated model operators for expressing component extraction and composition. This yielded a formalisation of model components, model extraction and model composition. All these notions are also currently being reflected in the Coq proof assistant, following the line of thought of our previous work around model and metamodel formalisation. This embedding provides us correct-by-construction pieces of executable code for the different model operations related to composition. For

[21] http://ptolemy.eecs.berkeley.edu/

instance model extraction and model composition are both proved to be terminating, the latter operation being in addition correct, as advocated by the main theorem. As we target a general purpose MDE-oriented framework, our work applies to any model, modeling language, application and is not restricted to some more-or-less implicit language context.

Yet, for the ease of experimentation, we have in a first step somehow restricted the possibilities of our composition framework. For instance, the notion of conformity, a notion at the heart of our formal description, has been temporarily weakened to take into account only instantiation constraints, disregarding any other model property (multiplicity, etc).

As future work, all these constraints should be enforced to achieve a fully-fledged formal model composition framework.

Furthermore, the interplay between model composition (where objects are replaced by others, assuming they have the same type) and sub-typing (where a single object may exhibit many types, due to duplication) needs to be clearly worked out in our framework.

This proposal is a preliminary mandatory step in the formalization of compositional formal verification technologies. We have tackled the formal composition of models from model fragments independently of the properties satisfied by the model fragments and the expected properties for the composite model. The next step in our work is to formalize the notion of model verification relying on several use case from simple static constraints such as typing or verification of OCL constraints satisfaction, to more dynamic properties such as deadlock freedom as proposed in the BIP framework. The expected result of our work is a framework to define compositional verification technologies and to prove the correctness of the associated verification tools.

## References


1. Aßmann, U.: Invasive software composition. Springer-Verlag New York Inc (2003)
2. Azurat, A.: Mechanization of invasive software composition in F-logic. In: Proceedings of the 2007 annual Conference on International Conference on Computer Engineering and Applications. pp. 89–94. World Scientific and Engineering Academy and Society (WSEAS) (2007)
3. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in BIP. In: Software Engineering and Formal Methods, 2006. SEFM 2006. Fourth IEEE International Conference on. pp. 3–12. IEEE (2006)
4. Bensalem, S., Bozga, M., Nguyen, T., Sifakis, J.: Compositional verification for component-based systems and application. Software, IET 4(3), 181–193 (2010)
5. Bernstein, P., Halevy, A., Pottinger, R.: A vision for management of complex models. ACM Sigmod Record 29(4), 55–63 (2000)
6. Boronat, A., Meseguer, J.: An algebraic semantics for mof. Formal Asp. Comput. 22(3-4), 269–296 (2010)
7. Clarke, S.: Extending standard UML with model composition semantics. Science of Computer Programming 44(1), 71–100 (2002)
8. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.: Maude: specification and programming in rewriting logic. Theoretical Computer Science 285(2), 187–243 (2002)

9. Fabro, M.D.D., Valduriez, P.: Towards the efficient development of model transformations using model weaving and matching transformations. Software and System Modeling 8(3), 305–324 (2009)
10. Giorgino, M., Strecker, M., Matthes, R., Pantel, M.: Verification of the Schorr-Waite algorithm - From trees to graphs. In: International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'10) (2010)
11. Heidenreich, F., Henriksson, J., Johannes, J., Zschaler, S.: On language-independent model modularisation. Transactions on Aspect-Oriented Software Development VI pp. 39–82 (2009)
12. Henriksson, J.: A Lightweight Framework for Universal Fragment Composition—with an application in the Semantic Web. Ph.D. thesis, PhD thesis, TU Dresden (January 2009)
13. Jeanneret, C.: An Analysis of Model Composition Approaches. Master's thesis, Ecole Polytechnique Fédérale de Lausanne (2007-2008)
14. Johannes, J.: Component-Based Model-Driven Software Development. Ph.D. thesis, vorgelegt an der Technischen Universität Dresden Fakultät Informatik (2011)
15. Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-Oriented Programming. In: Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP), Lecture Notes in Computer Science, vol. 1241, pp. 220–242. Springer (Jun 1997)
16. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. Journal of the ACM 42(4), 741–843 (1995)
17. Object Management Group, Inc.: Meta Object Facility (MOF) 2.0 Core Specification (Jan 2006), `http://www.omg.org/docs/formal/06-01-01.pdf`, final Adopted Specification
18. Picard, C., Matthes, R.: Coinductive graph representation : the problem of embedded lists. Electronic Communications of the EASST, Special issue Graph Computation Models, GCM'10 (2011)
19. Poernomo, I.: The meta-object facility typed. In: Haddad, H. (ed.) SAC. pp. 1845–1849. ACM (2006)
20. Poernomo, I.: Proofs-as-model-transformations. In: Vallecillo, A., Gray, J., Pierantonio, A. (eds.) ICMT. Lecture Notes in Computer Science, vol. 5063, pp. 214–228. Springer (2008)
21. Poernomo, I., Terrell, J.: Correct-by-construction model transformations from partially ordered specifications in coq. In: Dong, J.S., Zhu, H. (eds.) ICFEM. Lecture Notes in Computer Science, vol. 6447, pp. 56–73. Springer (2010)
22. Romero, J.R., Rivera, J.E., Durán, F., Vallecillo, A.: Formal and tool support for model driven engineering with maude. Journal of Object Technology 6(9), 187–207 (2007)
23. Thirioux, X., Combemale, B., Crégut, X., Garoche, P.L.: A Framework to Formalise the MDE Foundations. In: Paige, R., Bézivin, J. (eds.) International Workshop on Towers of Models (TOWERS). pp. 14–30. Zurich (Jun 2007)
24. Troya, J., Vallecillo, A.: Towards a rewriting logic semantics for atl. In: Tratt, L., Gogolla, M. (eds.) ICMT. Lecture Notes in Computer Science, vol. 6142, pp. 230–244. Springer (2010)
25. Xie, F., Browne, J.: Verified systems by composition from verified components. ACM SIGSOFT Software Engineering Notes 28(5), 277–286 (2003)