From subsets of model elements to submodels

Bernard Carré, Gilles Vanwormhoudt, Olivier Caron

Manuscript

The final publication is available at Springer via:
http://dx.doi.org/10.1007/s10270-013-0340-x

To cite this paper:
B. Carré, G. Vanwormhoudt, and O. Caron. From subsets of model elements to submodels, a characterization of submodels and their properties. Software and Systems Modeling, 14 (2) pp. 861–887, Springer, May 2015.

Software and Systems Modeling manuscript No.
(will be inserted by the editor)

From Subsets of Model Elements to Submodels

A Characterization of Submodels and their Properties

Bernard Carré[1], **Gilles Vanwormhoudt**[1,2], **Olivier Caron**[1]

[1] LIFL, UMR CNRS 8022
University of Lille
France - 59655 Villeneuve d'Ascq cedex
e-mail: {Bernard.Carre, Gilles.Vanwormhoudt, Olivier.Caron}@lifl.fr
[2] Institut TELECOM

The date of receipt and acceptance will be inserted by the editor

Abstract Model-Driven Engineering (MDE) generalized the status of models from documentation or MDA (Model Driven Architecture) modeling steps to full artifacts, members of a so-called structured "model space". We concentrate here on the submodel relationship which contributes a lot to this structuring effort. Many works and MDE practices resort to this notion and call for its precise characterization, which is the intent of this paper. A typical situation is model management through repositories. We start from the definition of a model as a set of model elements *plus* a set of dependency constraints that it asserts over these elements. This allows to isolate the notions of *closed*, *covariant* and *invariant* submodels. As a major result, we show that submodel transitivity can be guaranteed thanks to submodel invariance. This formalization offers keys to analyze operations which manipulate submodels. For example, we deeply study the operator which consists in extracting a model from another one, when selecting some subset of its elements. The same can be applied to many other model operations and the last part of the paper is dedicated to a synthesis on related works which could profit from this characterization. More practically, we show how the results were exploited in our Eclipse modeling environment.

Key words Submodel – Set-Theoretic Formalization – Model Extraction – Model Composition

1 Introduction

Model-Driven Engineering (MDE) generalized the status of models from documentation or MDA (Model Driven Architecture [1]) modeling steps to full first-class software objects, members of a so-called multi-dimensionnal "model space" [27]. This space is gradually gaining its own structure, thanks to proper internal relationships with their properties and rules, allowing rich composition and transformation operations. We concentrate here on the submodel relationship which contributes a lot to this structuring effort. Many works and MDE practices call for some submodel notion, for example:

- Incremental model construction (or editing) processes which gradually build and transform intermediate submodels up to the final required one [3,15].
- Model composition, through model assembly [16,14], AOM (Aspect Oriented Modeling) [6,38,50] or Template application [13,36], use model components as contributing submodels in the construction of the required model.
- Model querying, extraction and slicing [35,26,32] allow to present submodels of an overall model as simplified, more comprehensive and checkable model parts, when selecting model elements.

All these practices call for precise definition and properties of the submodel notion. Recurring questions are:

- How far a subset of model elements of some model m can form a submodel of m?
- What about the submodel relationship transitivity? The quest for transitivity is important because it guarantees qualities such as locality and modularity to the model space and its processing.
- Given a model m, how to produce a consistent submodel from the selection of any subset of its model elements? In the same vein, what about the transitivity, so the qualities, of this operation?

It is worth noting that, in most situations, submodels may range from full entire models to unspecified model parts, that is models which do not respect exactly the structure imposed by their metamodel. Figure 1 depicts a typical example of such "degenerated" models. Consider the model *inter* which is the common part of m

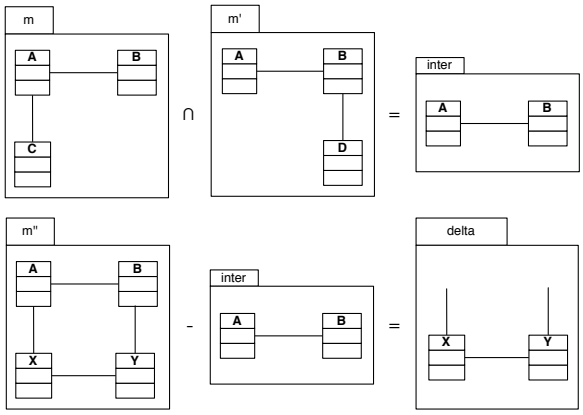

Fig. 1 Intersection and difference of models

and m'. This model could have been used in the construction of other models and model differencing allows to retrieve model fragments that were added. This process may produce models with "dangling edges" such as *delta* in the figure, when applied to m''. Model differencing is currently used in model versioning [3,48], collaborative design [46] and more generally model management [35].

It is the intent of this paper to offer an algebraic tool which helps in a better understanding and control of the submodel notion. It allows to cover and compare all the forms of submodels, being well-formed or not, in an homogeneous and consistent way. Before its formulation, we sketch in Section 2 a typical and motivating situation which does call for such submodels and their relationships, namely model management as offered by repositories [8,19,34].

Then the formalism is presented. As a starting point (Section 3), models are defined as sets of model elements *plus* the dependencies constraints that they assert over these elements, reflecting their proper structure. This formulation is simple but powerful enough to define models and submodels, being well-formed or not, and characterize their inclusion properties (Section 4). We isolate the concepts of *covariant* and *invariant* submodels, depending on the variation of the submodel structuring constraints compared to the overall ones. As a major result, submodel transitivity can be guaranteed thanks to submodel invariance.

This formalism offers keys to analyze MDE practices and operations which manipulate submodels. As an example, we present in Section 5 its application to the formalization of "model extraction". This operation is currently used in MDE practices and consists in extracting a submodel from another one when selecting a subset of its model elements. Thanks to the formalism, properties of this operation are systematically examined. This allows to precisely position the produced models as qualified submodels compared to the overall models under consideration.

At a much more practical level, Section 6 presents a concrete application which directly profits of the results. It is an Eclipse submodel engine that we use in our modeling environment in order to manage models and submodels issued from heterogeneous metamodels. The engine provides a set of core services which facilitate operations that manipulate submodels such as rich content-driven model querying, browsing, visualization in the large or model completion. The engine is extensible thanks to plugin-based style promoted by Eclipse. Adapters were developed for EMOF, UML Class and UML Collaboration diagrams. But it can easily support any other kind of model. The only thing to do is to write and plug the corresponding adapter, with respect to the framework requirements.

Finally (Section 7), related works are presented. Other works on the submodel notion are compared and related applications are examined, before concluding (Section 8).

2 Motivation

A typical situation where submodels are widely used is model management as offered by model repositories [8, 19,34]. Rationale of model repositories is to facilitate the storing of modeling design efforts and then their reuse. Indeed, creating models from scratch requires important design efforts. To be more effective, it is therefore more desirable to construct new models from existing ones. Intents of model repositories are mainly of two kinds:

1. Model capitalization and the constitution of "off-the-shelf " reusable models and model libraries [16,36,7].
2. Storing any modeling design effort. This facilitates incremental design, versioning, model sharing within design teams and finally traceability [35,3,46,48].

Intents of kind 1 call for entire model composition modes as those depicted by standard UML dependency relationships such as:

- *import*: which allows to use and refer to modeling elements of another model in the general case
- *merge*: whose intent is more accurately dedicated to model generalization/specialization
- *bind*: relates a model to "model templates" that were used in its construction.

These dependencies relate full well-formed models (with respect to their metamodel) to each other and trace how they compose in some way.

But model repositories allow to capitalize and reuse much more fine grained forms of modeling design efforts, following intents of kind 2. This calls for the management of not only full well-formed models but also unspecified model fragments. Samples are:

- simple (unstructured) sets of model ingredients

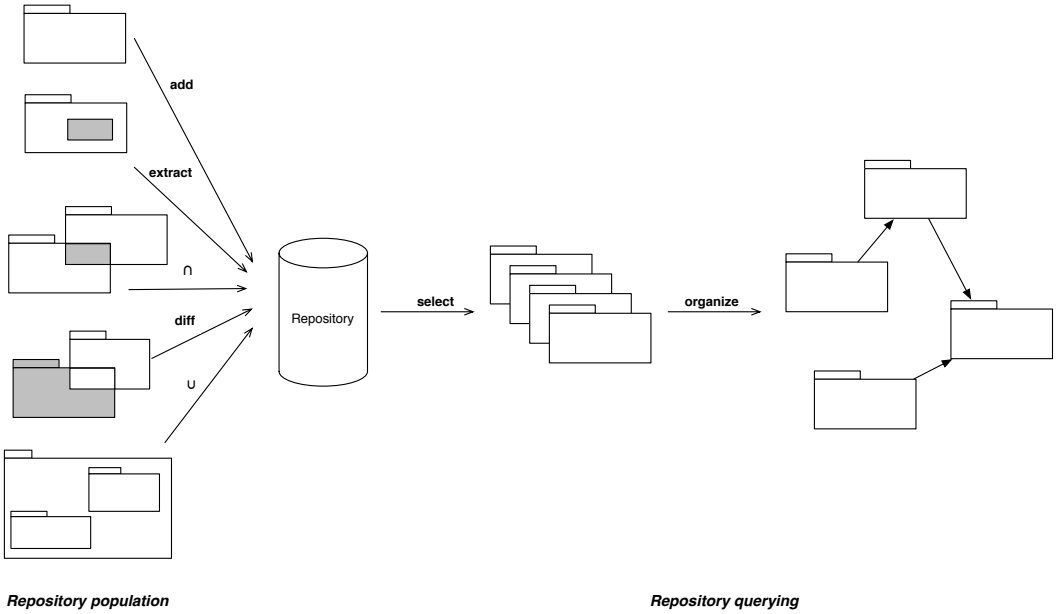

Fig. 2 Repository Population and Querying

– "model deltas" due to incremental design and versioning
– results of operations such as *intersection*, *difference* of models (Figure 1) and more primitively model element deleting
– more generally any intermediate model, being well-formed or not, envisaged in the project.

It is worth noting that such kinds of artifacts are frequently not well-formed, i.e they do not respect their metamodeling constraints, such as cardinalities or containment hierarchy ones. In particular this requires the handling of models with "dangling edges" [46], for example "delta" in Figure 1, which is the result of model difference. This operation is very similar to graph differencing and makes the representation of such models with graphs difficult, as identified in [33,48].

Model repositories have to mix and compare all these forms of models in their operation (Figure 2). Repositories must allow to register and classify model artifacts resulting from any design effort through population primitives and offer reusing facilities through rich querying operations such as:

– Retrieve ressources that include a specific set of model ingredients.
– Extract the (partial) model formed by some subset of its elements.
– Given a model m, possibly partial, retrieve models whose m is a (contributing) submodel.
– Conversely, retrieve registered model parts (contributions) of a given model.
– Organize the results thanks to inclusion properties.
– ...

All these model management operations lead to model spaces which call for systematic structuring through comparison of all the involved artifacts. The following offers a formalism for submodels and their corresponding inclusion properties which allows to understand and control underlying phenomena, illustrated by this typical situation.

3 Models

In the present formalization, a model is regarded as a set of model elements *plus* the dependency constraints that it asserts over these elements. We will see in the following sections that, for a subset of model elements of a model m to be considered as a submodel of m, it must respect these constraints. Before that, let us introduce this definition of a model and give a running example.

Notation Let $ModelElements$ being the set of all model elements of the modeling space under consideration. For any model m, we will note $\widetilde{m}$ the set of its model elements, so that: $\widetilde{m} \subseteq ModelElements$.

Definition 1 (Model)
A model is defined as a couple $m = (\widetilde{m}, \sqsubseteq_m)$ *such as:*

– $\widetilde{m} \subseteq ModelElements$ *is the set of model elements of* m
– $\sqsubseteq_m$: $\widetilde{m} \times \widetilde{m}$ *is a partial ordering relationship, local to* $\widetilde{m}$*, which translates the dependency constraints asserted by* m *on its model elements.*

Thanks to MDE technology, a modeling space can easily be determined by its metamodel which states the

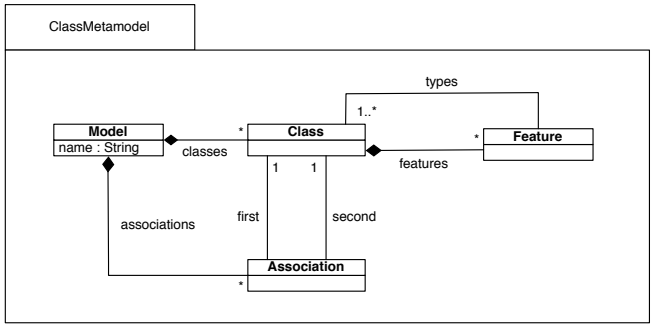

Fig. 3 Simplified Class Metamodel

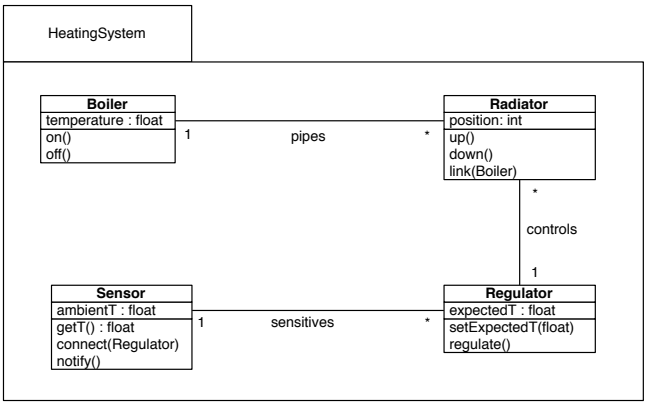

Fig. 4 Heating System

sets of model elements (*ModelElements*) and possible dependency constraints which can apply upon these elements. Take UML:

- When considering its whole modeling space, *ModelElements* would be the extension of the primitive `Element` root metaclass, and possible constraints would be determined by its overall metamodel.
- When considering particular diagrams such as *Class* or *Sequence* or any other ones, the sets of possibly engaged model elements (*ModelElements*) and dependencies will be delimited by the specific view (part) of the overall metamodel.

Consider the simplified *Class Metamodel* in Figure 3: $ModelElements = C \cup F \cup A$ where C, F, A correspond respectively to the sets of all *Classes*[1], *Features* (attributes or operations) and *Associations*. Then, for any model $m = (\widetilde{m}, \sqsubseteq_m)$ such as $\widetilde{m} \subseteq ModelElements$, possible dependency constraints are:

- Feature $f \in \widetilde{m} \cap F$ is a constituent of class $c \in \widetilde{m} \cap C$, noted $f \sqsubseteq_m c$ (see the *features* association between *Class* and *Feature* in the figure).
- Feature $f \in \widetilde{m} \cap F$ (attribute or operation) depends on type $t \in \widetilde{m} \cap C$, noted $f \sqsubseteq_m t$ (see the *types* association between *Feature* and *Class*).
- Association $a \in \widetilde{m} \cap A$ relates to some end (*first* or *second* in the figure) $c \in \widetilde{m} \cap C$, noted $a \sqsubseteq_m c$.

Finally, $\sqsubseteq_m$ being a preorder, it is reflective so that we must have[2]: $\forall x \in \widetilde{m}, x \sqsubseteq_m x$.

For example, the model *HeatingSystem* in Figure 4, which will be used as a running example throughout the paper, can be represented as $HeatingSystem = (\widetilde{HeatingSystem}, \sqsubseteq_{HeatingSystem})$ where:

- $\widetilde{HeatingSystem}$ = {*Boiler, temperature, float, on, off, pipes, Radiator, position, int, up, down, link, controls, Regulator, expectedT, setExpectedT, regulate, sensitives, Sensor, ambientT, getT, connect, notify*}
- $\sqsubseteq_{HeatingSystem}$:
  - $temperature \sqsubseteq_{HeatingSystem} float$
  - $temperature \sqsubseteq_{HeatingSystem} Boiler$
  - $on \sqsubseteq_{HeatingSystem} Boiler$
  - $off \sqsubseteq_{HeatingSystem} Boiler$
  - $pipes \sqsubseteq_{HeatingSystem} Boiler$
  - $pipes \sqsubseteq_{HeatingSystem} Radiator$
  - $position \sqsubseteq_{HeatingSystem} int$
  - $position \sqsubseteq_{HeatingSystem} Radiator$
  - $up \sqsubseteq_{HeatingSystem} Radiator$
  - $down \sqsubseteq_{HeatingSystem} Radiator$
  - $link \sqsubseteq_{HeatingSystem} Boiler$
  - $link \sqsubseteq_{HeatingSystem} Radiator$
  - $controls \sqsubseteq_{HeatingSystem} Radiator$
  - $controls \sqsubseteq_{HeatingSystem} Regulator$
  - $expectedT \sqsubseteq_{HeatingSystem} float$
  - $expectedT \sqsubseteq_{HeatingSystem} Regulator$
  - $setExpectedT \sqsubseteq_{HeatingSystem} float$
  - $setExpectedT \sqsubseteq_{HeatingSystem} Regulator$
  - $regulate \sqsubseteq_{HeatingSystem} Regulator$
  - $sensitives \sqsubseteq_{HeatingSystem} Regulator$
  - $sensitives \sqsubseteq_{HeatingSystem} Sensor$
  - $ambientT \sqsubseteq_{HeatingSystem} float$
  - $ambientT \sqsubseteq_{HeatingSystem} Sensor$
  - $getT \sqsubseteq_{HeatingSystem} float$
  - $getT \sqsubseteq_{HeatingSystem} Sensor$
  - $connect \sqsubseteq_{HeatingSystem} Regulator$
  - $connect \sqsubseteq_{HeatingSystem} Sensor$
  - $notify \sqsubseteq_{HeatingSystem} Sensor$

It is important to note that the *model* definition is general enough to represent full models, such as the preceding one, but also any partial ones as far as a set of model elements and a set of dependency constraints over these elements are provided.

[1] To simplify, primitive data types are considered as classes, their typing function being approximated by the association between *Feature* and *Class*, under the *types* role. This has no consequence in this paper.

[2] For the sake of simplicity, these systematic reflective constraints will not be listed in the examples.

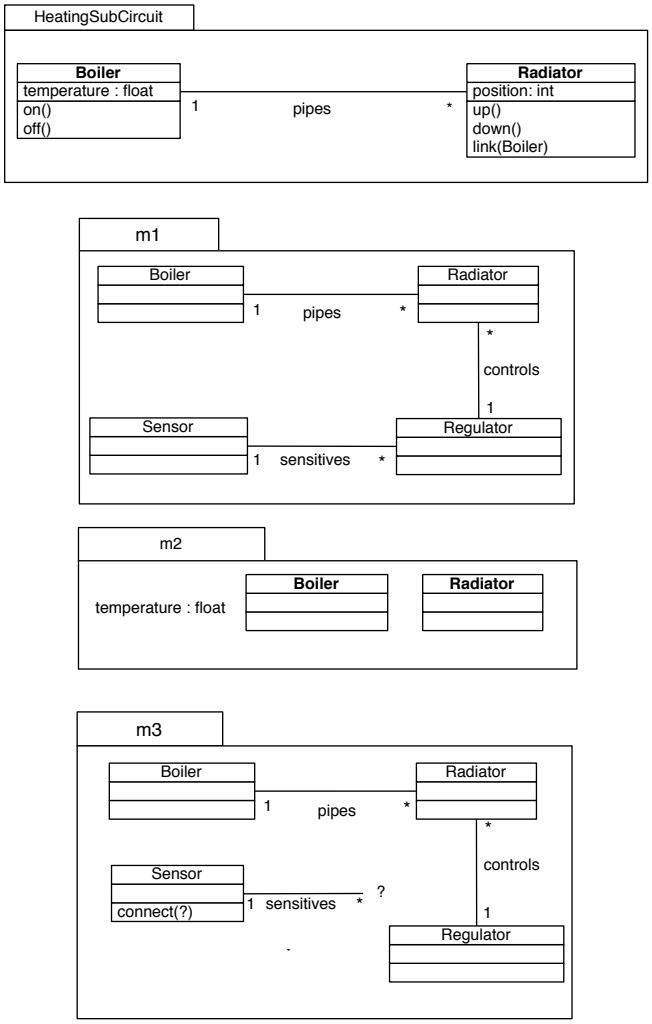

Fig. 5 Four Partial Models of the Metamodel of Figure 3

Figure 5 shows samples of partial models, which intuitively raise the issues. Model *HeatingSubCircuit* can be defined as $(\widetilde{HeatingSubCircuit}, \sqsubseteq_{HeatingSubCircuit})$:

- $\widetilde{HeatingSubCircuit}$ ={*temperature, float, Boiler, on, off, pipes, Radiator, position, int, up, down, link*}
- $\sqsubseteq_{HeatingSubCircuit}$:
  - $temperature \sqsubseteq_{HeatingSubCircuit} float$
  - $temperature \sqsubseteq_{HeatingSubCircuit} Boiler$
  - $on \sqsubseteq_{HeatingSubCircuit} Boiler$
  - $off \sqsubseteq_{HeatingSubCircuit} Boiler$
  - $pipes \sqsubseteq_{HeatingSubCircuit} Boiler$
  - $pipes \sqsubseteq_{HeatingSubCircuit} Radiator$
  - $position \sqsubseteq_{HeatingSubCircuit} int$
  - $position \sqsubseteq_{HeatingSubCircuit} Radiator$
  - $up \sqsubseteq_{HeatingSubCircuit} Radiator$
  - $down \sqsubseteq_{HeatingSubCircuit} Radiator$
  - $link \sqsubseteq_{HeatingSubCircuit} Boiler$
  - $link \sqsubseteq_{HeatingSubCircuit} Radiator$

This model appears to be a full well-formed submodel of the overall *HeatingSystem* model. The same applies to the model *m1* which is unaware of classes constituent details but concentrates on their relationships. Model $m1$ is defined as:

- $\widetilde{m1}$ ={*Boiler, Radiator, pipes, Regulator, controls, Sensor, sensitives*}
- $\sqsubseteq_{m1}$:
  - $pipes \sqsubseteq_{m1} Boiler$
  - $pipes \sqsubseteq_{m1} Radiator$
  - $controls \sqsubseteq_{m1} Radiator$
  - $controls \sqsubseteq_{m1} Regulator$
  - $sensitives \sqsubseteq_{m1} Regulator$
  - $sensitives \sqsubseteq_{m1} Sensor$

Models *m2* and *m3* are samples of incompletely specified ones due, for example, to intermediate models envisaged in the construction of the Heating System, or to the selection of some part of its resulting model:

- $m2 = (\widetilde{m2}, \sqsubseteq_{m2})$
  - $\widetilde{m2}$ ={*temperature, float, Boiler, Radiator*}
  - $\sqsubseteq_{m2}$:
    - $temperature \sqsubseteq_{m2} float$
- $m3 = (\widetilde{m3}, \sqsubseteq_{m3})$
  - $\widetilde{m3}$ ={*Boiler, Radiator, pipes, Regulator, controls, Sensor, sensitives, connect*}
  - $\sqsubseteq_{m3}$:
    - $pipes \sqsubseteq_{m3} Boiler$
    - $pipes \sqsubseteq_{m3} Radiator$
    - $controls \sqsubseteq_{m3} Radiator$
    - $controls \sqsubseteq_{m3} Regulator$
    - $sensitives \sqsubseteq_{m3} Sensor$
    - $connect \sqsubseteq_{m3} Sensor$

4 Submodels and their properties

Based on the preceding model definition, this section presents the concepts of *closed*, *covariant* and *invariant submodels*. These nested concepts allow to precisely qualify the inclusion relationships that may apply between two models and progressively obtain submodel transitivity. As a starting point the notion of closed submodel focusses on the sets of model elements. The notion is not transitive but an intermediate property ("Bound Closure"), with interesting qualities, can be stated. Then, the notions of covariant and invariant submodels allow to compare models depending on the variation of their dependencies constraints. It will be shown that the transitivity of the submodel notion can be guaranteed thanks to invariance.

4.1 Closed Submodel of a Model

Definition 2 (Closed Subset of a Model)
Let $m = (\widetilde{m}, \sqsubseteq_m)$ a model, $s \subseteq ModelElements$ a set of model elements, we will say that s is closed in m iff:

1. $s \subseteq \widetilde{m}$

2. s is transitively closed by the $\sqsubseteq_m$ relationship, that is: $closure_{\sqsubseteq_m}(s) = s$

Where $closure_{\sqsubseteq_m}(s) = \{x \in \widetilde{m} \mid \exists y \in s, y \sqsubseteq_m^* x\}$, that is the set of model elements of m whose elements of s transitively depend, with respect to the dependency constraints asserted by m.

And, by extension:

Definition 3 (Closed Submodel of a Model) *We will say that a model $m = (\widetilde{m}, \sqsubseteq_m)$ is a closed submodel of a model $m' = (\widetilde{m'}, \sqsubseteq_{m'})$ (or simply m is closed in m'), iff*

- $\widetilde{m} \subseteq \widetilde{m'}$
- *$\widetilde{m}$ is closed in m', that is: $closure_{\sqsubseteq_{m'}}(\widetilde{m}) = \widetilde{m}$*

For example, consider Figure 6 which shows possible parts of the *HeatingSystem* model. Following are their formulation as $m = (\widetilde{m}, \sqsubseteq_m)$:

- $m0 = (\widetilde{m0}, \sqsubseteq_{m0})$
  - $\widetilde{m0}$ =$\{$*temperature, float*$\}$=s
  - $\sqsubseteq_{m0}$:
    - $temperature \sqsubseteq_{m0} float$
- $m1 = (\widetilde{m1}, \sqsubseteq_{m1})$
  - $\widetilde{m1}$ =$\{$*temperature, float, Boiler, on, off, Radiator, up, down*$\}$
  - $\sqsubseteq_{m1}$:
    - $temperature \sqsubseteq_{m1} float$
    - $on \sqsubseteq_{m1} Boiler$
    - $off \sqsubseteq_{m1} Boiler$
    - $up \sqsubseteq_{m1} Radiator$
    - $down \sqsubseteq_{m1} Radiator$

We have:

- $m0$ (or s) is closed in $m1$
- $m1$ is closed in *HeatingSubCircuit* and *HeatingSystem*:

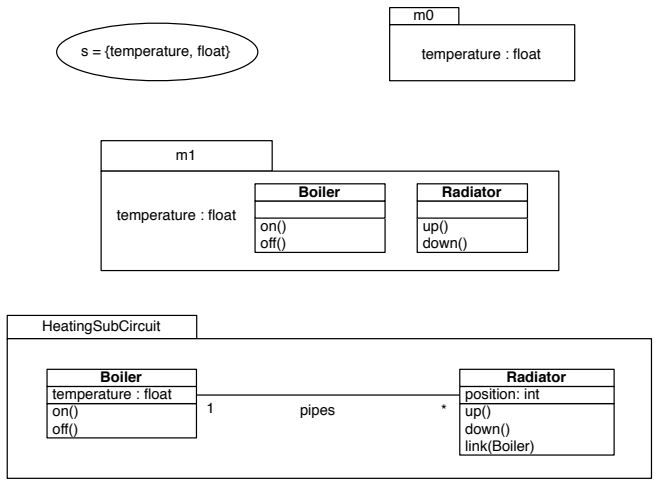

Fig. 6 Closed/Not Closed Submodels

- *HeatingSubCircuit* is closed in *HeatingSystem*

But $m0$ is not closed in *HeatingSubCircuit* (nor *HeatingSystem*) due to the dependency between *temperature* and *Boiler* in this model:
$closure_{\sqsubseteq_{HeatingSubCircuit}}(\{temperature, float\})$
$= \{temperature, float, Boiler\} \neq \widetilde{m0}$.

As expected, closure is not transitive: $m0$ is closed in $m1$, $m1$ is closed in *HeatingSubCircuit*, but $m0$ is not closed in *HeatingSubCircuit*. Though, a weaker property can be demonstrated:

Property 1 (Bound Closure)

Let two models $m = (\widetilde{m}, \sqsubseteq_m)$, $m' = (\widetilde{m'}, \sqsubseteq_{m'})$ and a subset $s \subseteq \widetilde{m}$,
s is closed in m and m is closed in m'
$\Rightarrow closure_{\sqsubseteq_{m'}}(s) \subseteq \widetilde{m}$

Proof (by contradiction) :
$closure_{\sqsubseteq_{m'}}(s) \subseteq \widetilde{m}$ iff: $\forall x \in s, \nexists y \in \widetilde{m'} \setminus \widetilde{m}, x \sqsubseteq_{m'} y$.
Suppose: $\exists y \in \widetilde{m'} \setminus \widetilde{m}, x \sqsubseteq_{m'} y\, (y \notin \widetilde{m})$
$x \in s, s \subseteq \widetilde{m} \Rightarrow x \in \widetilde{m}$
$x \sqsubseteq_{m'} y \Rightarrow y \in closure_{\sqsubseteq_{m'}}(\widetilde{m})$
but m is closed in $m' \Rightarrow closure_{\sqsubseteq_{m'}}(\widetilde{m}) = \widetilde{m}$
so the contradiction:$y \in \widetilde{m}$. □

This property is a step towards submodel transitivity as we will see later. But far beyond this step, it must be noted at this stage that this property is interesting on its own. Indeed it characterizes locality and modularity qualities of the notion of closed submodels. The following situations exhibit this.

See Figure 7 and the overall model m'. Then consider any submodel of m' such as m whose set of model elements is closed in m'. As a consequence m appears to be a limit for the closure in m' of any of its subsets such as s.

For example (Figure 6), *HeatingSubCircuit*, which is closed in *HeatingSystem*, determines a limit for the closure in this overall model of any of its subsets of model elements such as:

- $\widetilde{m1}$ which is closed in it
- but also $\widetilde{m0}$ which is not, although $closure_{\sqsubseteq_{HeatingSystem}}(\widetilde{m0})$ is bound by *HeatingSubCircuit*:
  $closure_{\sqsubseteq_{HeatingSystem}}(\widetilde{m0})$
  $= closure_{\sqsubseteq_{HeatingSystem}}(\{temperature, float\})$
  $= \{temperature, float, Boiler\}$
  $\subseteq \widetilde{HeatingSubCircuit}$.

This is not true when m is not closed in m' as depicted in Figure 8. Since m is not closed in m', the closure in m' of any of its subsets, let s, cannot be bound by m. Indeed the closure of s in m' may expand from s up to $closure_{\sqsubseteq_{m'}}(\widetilde{m})$, this is trivial.

For example, consider the situation shown in Figure 9:

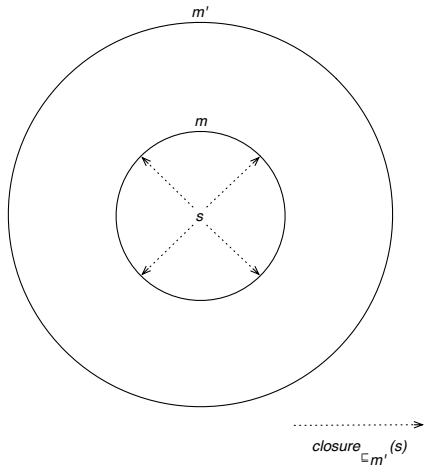

Fig. 7 Bound Closed Subset

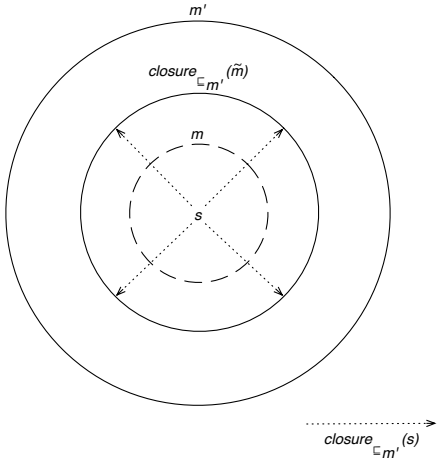

Fig. 8 Unbound Subsets

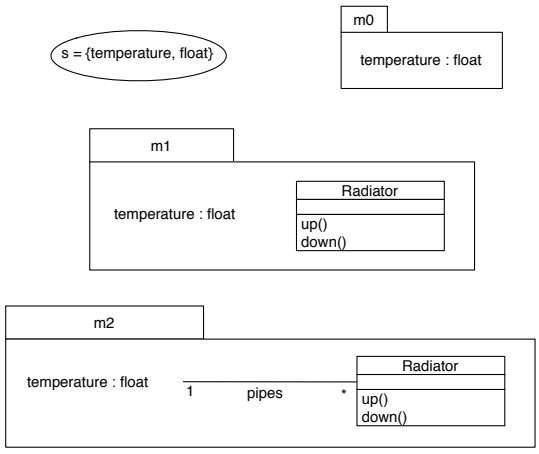

Fig. 9 Unbound Subsets Example

- $m0 = (\widetilde{m0}, \sqsubseteq_{m0})$
  - $\widetilde{m0}$ =s=$\{temperature,\ float\}$
  - $\sqsubseteq_{m0}$:
    - $temperature \sqsubseteq_{m0} float$
- $m1 = (\widetilde{m1}, \sqsubseteq_{m1})$
  - $\widetilde{m1}$ =$\{temperature,\ float,\ Radiator,\ up,\ down\}$
  - $\sqsubseteq_{m1}$:
    - $temperature \sqsubseteq_{m1} float$
    - $up \sqsubseteq_{m1} Radiator$
    - $down \sqsubseteq_{m1} Radiator$
- $m2 = (\widetilde{m2}, \sqsubseteq_{m2})$
  - $\widetilde{m2}$ =$\{temperature,\ float,\ pipes,\ Radiator,\ up,\ down\}$
  - $\sqsubseteq_{m2}$:
    - $temperature \sqsubseteq_{m2} float$
    - $pipes \sqsubseteq_{m2} Radiator$
    - $up \sqsubseteq_{m2} Radiator$
    - $down \sqsubseteq_{m2} Radiator$

We have:

- $s = \widetilde{m0}$ closed in $m1$ and $m2$
- $m1$ closed in $m2$

But $m2$ is not closed in $HeatingSystem$ (nor $HeatingSubCircuit$):
$closure_{\sqsubseteq_{HeatingSystem}}(\widetilde{m2})$
$= closure_{\sqsubseteq_{HeatingSystem}}(\{temperature, float, pipes, Radiator, up, down\})$
$= \{temperature, float, pipes, Radiator, up, down, Boiler\}$
$\neq \widetilde{m2}$.

So that $m2$ does not determine a limit for the closure in $HeatingSystem$ of its subsets $\widetilde{m0}$ and $\widetilde{m1}$, indeed:

- $closure_{\sqsubseteq_{HeatingSystem}}(\widetilde{m0})$
  $= \{temperature, float, Boiler\} \not\subseteq \widetilde{m2}$
- $closure_{\sqsubseteq_{HeatingSystem}}(\widetilde{m1})$
  $= \{temperature, float, Radiator, up, down, Boiler\}$
  $\not\subseteq \widetilde{m2}$.

On the contrary, as shown above, $HeatingSubCircuit$ remains a limit for the closure in $HeatingSystem$ of these new subset examples $\widetilde{m1}$ and $\widetilde{m2}$.

4.2 Covariant Submodels

The notion of *closed submodels* focused on the completeness (the closure) of their sets of model elements relatively to any overall model. Though it only permits to test whether any model m can be considered as a submodel of surrounding ones, based on its set of model elements dimension. The modeling structure of m and the question of its conformance with the overall model's one are not taken into account. It is the intent of the present section, which introduces the notion of *covariant submodels* and the following one on *invariant submodels*, to characterize this phenomenon.

Let a model $m = (\widetilde{m}, \sqsubseteq_m)$. Following the *closed submodel* notion, any subset of model elements of $\widetilde{m}$ associated to some combination of the structuring constraints

imposed by their metamodel can be considered as a submodel of m.

For example, consider Figure 10 which adds to Figure 6 the models $m2$ and $m2'$ as alternative extensions of $m1$:

- $m2 = (\widetilde{m2}, \sqsubseteq_{m2})$
  - $\widetilde{m2}$ ={*Boiler, temperature, float, on, off, Radiator, up, down*}
  - $\sqsubseteq_{m2}$:
    - $temperature \sqsubseteq_{m2} float$
    - $temperature \sqsubseteq_{m2} Boiler$
    - $on \sqsubseteq_{m2} Boiler$
    - $off \sqsubseteq_{m2} Boiler$
    - $up \sqsubseteq_{m2} Radiator$
    - $down \sqsubseteq_{m2} Radiator$
- $m2' = (\widetilde{m2'}, \sqsubseteq_{m2'})$
  - $\widetilde{m2'}$ ={*Boiler, temperature, float, on, off, Radiator, up, down*}
  - $\sqsubseteq_{m2'}$:
    - $temperature \sqsubseteq_{m2'} float$
    - $temperature \sqsubseteq_{m2'} Radiator$
    - $on \sqsubseteq_{m2'} Boiler$
    - $off \sqsubseteq_{m2'} Boiler$
    - $up \sqsubseteq_{m2'} Radiator$
    - $down \sqsubseteq_{m2'} Radiator$

Note that $m2$ and $m2'$ have the same set of model elements ($\widetilde{m2} = \widetilde{m2'}$). The only difference is the attribution of *temperature* either to *Boiler* or *Radiator* in the respective dependency constraints of $m2$ and $m2'$. Both submodels are closed in the $HeatingSubCircuit$ model. But $m2$ conforms to this model, contrary to $m2'$, which attributes *temperature* to *Radiator* rather than *Boiler*, as specified in $HeatingSubCircuit$.

This leads to take into account the dependency constraints of the candidate submodels in order to test whether they also respect the modeling structure of the overall model under consideration. So the following concept of *covariant submodel*.

Definition 4 (Covariant Submodel)
Let $m = (\widetilde{m}, \sqsubseteq_m)$ and $m' = (\widetilde{m'}, \sqsubseteq_{m'})$ two models, we say that m is a covariant submodel of m' (or more simply m is covariant in m') iff:

- *m is closed in m'*
- *and the dependency constraints of m are covariant with the m' ones, that is: $\forall x, y \in \widetilde{m}, x \sqsubseteq_m y \Rightarrow x \sqsubseteq_{m'} y$.*

An equivalent formulation can be done using the graph of the relation $\sqsubseteq_m$. For any model $m = (\widetilde{m}, \sqsubseteq_m)$, let us note $Gr(m)$ this graph: $Gr(m) = \{(x, y) \mid x, y \in \widetilde{m}, x \sqsubseteq_m y\}$. Then the other formulation:

Definition 5 (equivalent to Definition 4)
Let $m = (\widetilde{m}, \sqsubseteq_m)$ and $m' = (\widetilde{m'}, \sqsubseteq_{m'})$ two models, we say that m is a covariant submodel of m' (or more simply m is covariant in m') iff:

- *m is closed in m'*
- *and the dependency constraints of m are covariant with the m' ones, that is: $Gr(m) \subseteq Gr(m')$*

Figure 11 summarizes the *closed* and *covariant* relationships that apply to the models of Figure 10:

- $m0$ is a covariant submodel of $m1$, but $m1$ is not covariant in $m0$: $\widetilde{m1} \not\subseteq \widetilde{m0}$.
- $m1$ is a covariant submodel of $m2$, but $m2$ is not covariant in $m1$. Indeed, $m2$ is closed in $m1$ but $Gr(m2) \not\subseteq Gr(m1)$:
  - $Gr(m1)$ ={*(temperature,float), (on,Boiler), (off,Boiler), (up,Radiator), (down,Radiator)*}
  - $Gr(m2)$ ={*(temperature,float), (on,Boiler), (off,Boiler), (up,Radiator), (down,Radiator), (temperature,Boiler)*}
  - $(temperature, Boiler) \notin Gr(m1)$.
- $m1$ is a covariant submodel of $m2'$, but $m2'$ is not covariant in $m1$ for the same reason.
- $m2$ and $m2'$ are mutually closed (more, $\widetilde{m2} = \widetilde{m2'}$) but not covariant in each other, because of their structuring constraints:
  - $Gr(m2')$ ={*(temperature,float), (on,Boiler), (off,Boiler), (up,Radiator), (down,Radiator), (temperature,Radiator)*}
  - $Gr(m2) \not\subseteq Gr(m2')$ $((temperature, Boiler) \notin Gr(m2'))$ and $Gr(m2') \not\subseteq Gr(m2)$ $((temperature, Radiator) \notin Gr(m2))$.
- $m2$ and $m2'$ are closed in $HeatingSubCircuit$ but $m2$ is also covariant in $HeatingSubCircuit$ contrary to $m2'$, $Gr(m2') \not\subseteq Gr(HeatingSubCircuit)$: $(temperature, Radiator) \in Gr(m2')$ but $(temperature, Radiator) \notin Gr(HeatingSubCircuit)$.
- and finally $m1$ is covariant in $HeatingSubCircuit$ contrary to $m0$ which is not closed in this model, though $Gr(m0) \subseteq Gr(HeatingSubCircuit)$.

Note that the latter case ($m0$ covariant in $m1$, $m1$ covariant in $HeatingSubCircuit$, but $m0$ not covariant in $HeatingSubCircuit$) is a counter-example which proves, as expected, that submodel covariance is not transitive due to closure non-transitivity.

4.3 Invariant Submodels

The notion of *invariant submodel* is the cornerstone of this paper which will allow submodel transitivity. Let us introduce the problem intuitively. Thanks to the notions of *closed* and *covariant* submodels, it was possible to test whether models were included in an overall one in some way. The point is that some of these submodels appear to be "stable", when considering their structure, relatively to the overall model contrary to other ones.

For example, consider Figure 11 again. Models $m1$ and $m2$ have the same subset of model elements of $HeatingSubCircuit$ and are covariant in this model. Though, one

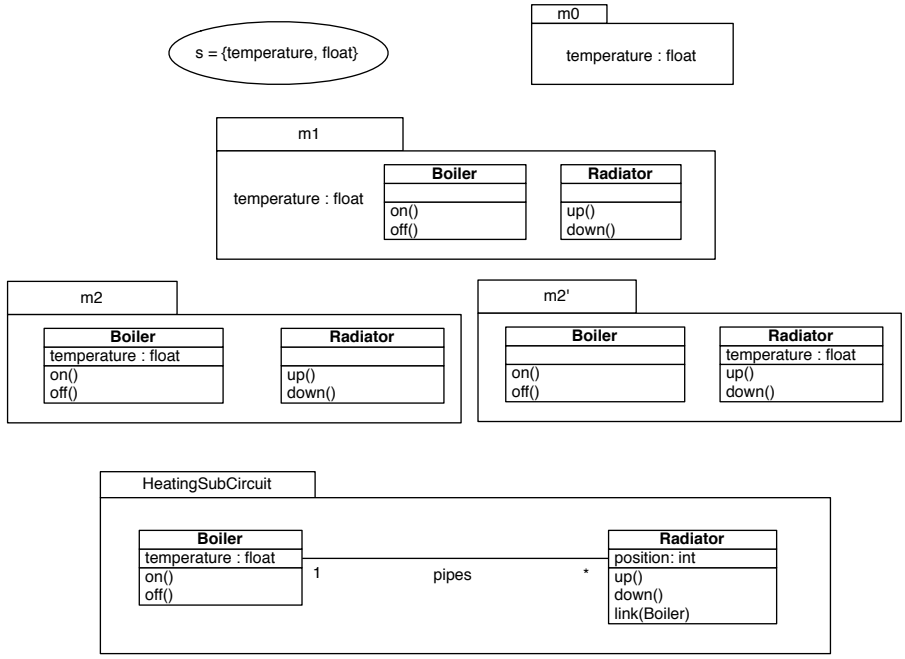

Fig. 10 Covariant/Not Covariant Submodels

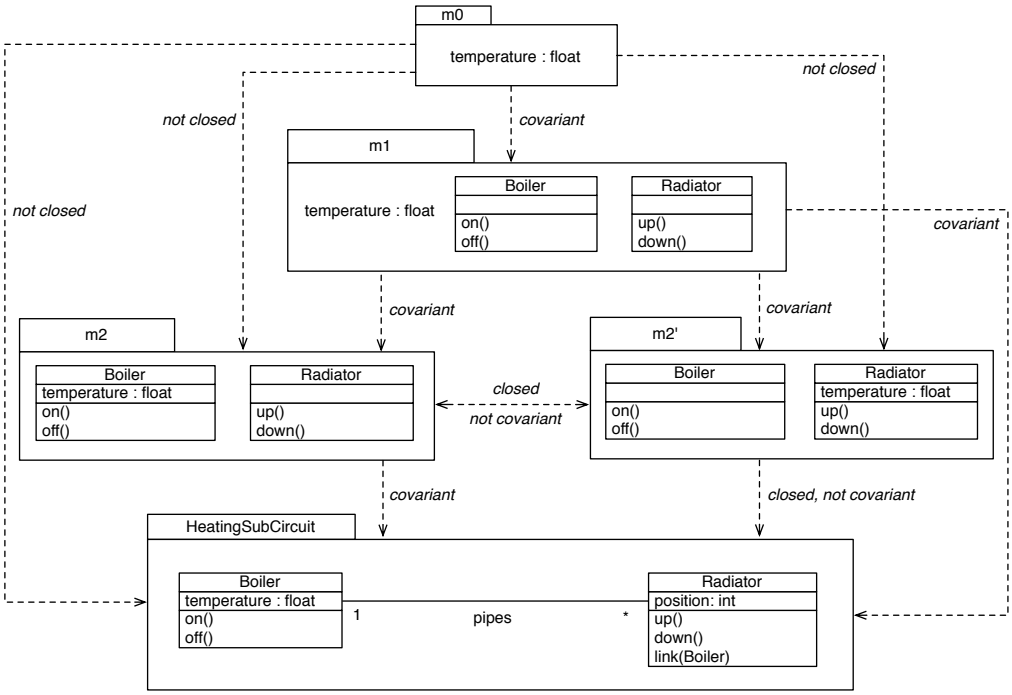

Fig. 11 Closed/Not Closed, Covariant/Not Covariant Submodels

can see that the structure of $m2$ is stable in $HeatingSubCircuit$ contrary to that of $m1$. Indeed the structuring constraints imposed by $m2$ on this common subset of model elements is the same in *HeatingSubCircuit* whereas those of $m1$ were under-specified.

The notion of *invariant submodel* captures this situation and characterizes the phenomenon.

Definition 6 (Invariant Submodel)
Let: $m = (\widetilde{m}, \sqsubseteq_m)$ and $m' = (\widetilde{m'}, \sqsubseteq_{m'})$ two models, we say that m is an invariant submodel of m' (or more simply m is invariant in m') iff:

- *m is closed in m'*
- *and the dependency constraints of m are invariant with the m' ones, that is: $\forall x, y \in \widetilde{m}, x \sqsubseteq_{m'} y \Leftrightarrow x \sqsubseteq_m y$*

Similarly to *covariant submodels*, an equivalent definition can be done using the notion of *graph of the relation* $\sqsubseteq_m$, with the following notation.

Notation Let a model $m = (\widetilde{m}, \sqsubseteq_m)$, for any subset s of its model elements ($s \subseteq \widetilde{m}$), we will note $Gr(m)/s$ the restriction of $Gr(m)$ to s, that is: $Gr(m)/s = \{(x, y) \mid x, y \in s, x \sqsubseteq_m y\}$

Then the other definition of *Invariant Submodel*:

Definition 7 (equivalent to Definition 6)
Let $m = (\widetilde{m}, \sqsubseteq_m)$ and $m' = (\widetilde{m'}, \sqsubseteq_{m'})$ two models, we say that m is an invariant submodel of m' (or more simply m is invariant in m') iff:

- *m is closed in m'*
- *and the dependency constraints of m are invariant with the m' ones, that is: $Gr(m')/\widetilde{m} = Gr(m)$*

For example, Figure 12 is an extension of Figure 11 by taking into account the notion of *Invariant Submodel*. We have:

- $m2$ is invariant in $HeatingSubCircuit$
- contrary to $m1$: the constraint $temperature \sqsubseteq_{m1} Boiler$ does not exist in $m1$.

Note also that:

- $m0$ is not invariant in $HeatingSubCircuit$: $m0$ is not closed in this model (*Boiler* is missing) so that the constraint $temperature \sqsubseteq_{m0} Boiler$ should not apply.
- $m2'$ is not invariant in $HeatingSubCircuit$: the constraint $temperature \sqsubseteq_{HeatingSubCircuit} Boiler$ was "replaced" by $temperature \sqsubseteq_{m2'} Radiator$, that is why $m2'$ was not covariant in this model:
  $Gr(HeatingSubCircuit)/\widetilde{m2'} = \{$*(temperature,float), (on,Boiler), (off,Boiler), (up,Radiator), (down,Radiator), (temperature,Boiler)*$\}$
  $\neq Gr(m2')$

Note that *submodel invariance* implies *submodel covariance*, which is trivial.

Property 2 Let two models m and m',
m is invariant in $m' \Rightarrow m$ is covariant in m'

Proof For m to be covariant in m':

- m must be closed in m', that is guaranteed by definition
- and the dependency constraints of m must be covariant with the ones of m', indeed:
  $Gr(m')/\widetilde{m} = Gr(m)$
  $\Rightarrow Gr(m) \subseteq Gr(m')/\widetilde{m} \subseteq Gr(m')$ □

Preceding model $m2$ is an example, $m0$ and $m2'$ are counter-examples:

- $m2$ is invariant so covariant in $HeatingSubCircuit$.
- $m0$ which is not closed in $HeatingSubCircuit$ cannot be covariant nor invariant (a fortiori) in this model.
- $m2'$ is not invariant in $HeatingSubCircuit$ because it is not covariant (though closed) in this model.

More generally, models which are not covariant (even more, not closed) in another one have no chance to be invariant in it.

But this time, *submodel transitivity* is found, thanks to the invariance of the dependency constraints. Before the demonstration, take a few intuitive examples to be convinced (see Figure 12):

- $m2$ is invariant in $HeatingSubCircuit$ which itself is invariant in $HeatingSystem$, and indeed $m2$ is invariant in $HeatingSystem$
- but $m1$ which is not invariant in $m2$, is no more invariant in these surrounding models
- the same applies one level below: $m0$, which is invariant in $m1$, is not invariant in $m2$ (nor in the preceding surrounding models, a fortiori), because $m1$ is not.

Here is the property:

Property 3 (Invariance Transitivity)
Let three models $m = (\widetilde{m}, \sqsubseteq_m)$, $m' = (\widetilde{m'}, \sqsubseteq_{m'})$ and $m'' = (\widetilde{m''}, \sqsubseteq_{m''})$,
m is invariant in m' and m' is invariant in m''
$\Rightarrow m$ is invariant in m''

Proof : m is invariant in m'' iff:

I. m is closed in m''.
II. The dependency constraints of m are invariant with the m'' ones.

Proof of I.
m is closed in m'' iff:

1. $\widetilde{m} \subseteq \widetilde{m''}$. Indeed:
   - m is invariant (so closed) in $m' \Rightarrow \widetilde{m} \subseteq \widetilde{m'}$.
   - m' is invariant (so closed) in $m'' \Rightarrow \widetilde{m'} \subseteq \widetilde{m''}$.
2. $closure_{m''}(\widetilde{m}) = \widetilde{m}$, that is:
   $\forall x \in \widetilde{m}, \nexists y \in \widetilde{m''} \setminus \widetilde{m}, x \sqsubseteq_{m''} y$.
   By contradiction, suppose:
   $\exists y \in \widetilde{m''} \setminus \widetilde{m}, x \sqsubseteq_{m''} y \, (y \notin \widetilde{m})$
   then, two cases must be considered: $y \in \widetilde{m'}$ or not (see diagram in Figure 13)

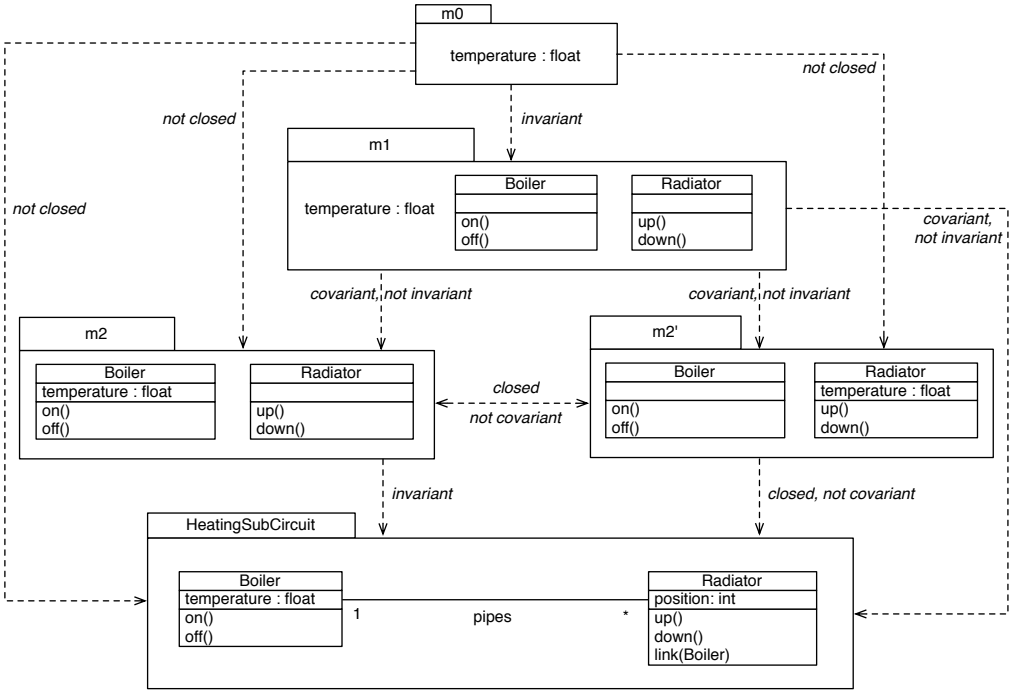

Fig. 12 Closed/Not Closed, Covariant/Not Covariant, Invariant/Not Invariant Submodels

(a) $y \in \widetilde{m'}$
- $x \in \widetilde{m} \Rightarrow x \in \widetilde{m'}$. Indeed by assumption m is invariant in m', so closed in $m' \Rightarrow \widetilde{m} \subseteq \widetilde{m'}$.
- So that we have: $x, y \in \widetilde{m'}$ with (by supposition) $x \sqsubseteq_{m''} y$. Since m' is invariant in m'', $x \sqsubseteq_{m''} y \Rightarrow x \sqsubseteq_{m'} y$. Then: $x \in \widetilde{m}$ and $x \sqsubseteq_{m'} y \Rightarrow y \in closure_{\sqsubseteq_{m'}}(\widetilde{m})$.
- But m being closed in m', $closure_{\sqsubseteq_{m'}}(\widetilde{m}) = \widetilde{m}$
- So the contradiction: $y \in \widetilde{m}$

(b) $y \notin \widetilde{m'}$ $(y \in \widetilde{m''} \setminus \widetilde{m'})$
This cannot occur:
- m being closed in m' which itself is closed in m'', Property 1 (*Bound Closure*) applies: $closure_{m''}(\widetilde{m}) \subseteq \widetilde{m'}$.
- So the contradiction: $x \in \widetilde{m}$ and $x \sqsubseteq_{m''} y$ (supposition) $\Rightarrow y \in closure_{m''}(\widetilde{m}) \subseteq \widetilde{m'} \Rightarrow y \in \widetilde{m'}$.

Proof of II.
The dependency constraints of m are invariant with the m'' ones, that is: $\forall x, y \in \widetilde{m}, x \sqsubseteq_{m''} y \Leftrightarrow x \sqsubseteq_m y$. This can be demonstrated in two steps:

1. $\forall x, y \in \widetilde{m}, x \sqsubseteq_m y \Rightarrow x \sqsubseteq_{m''} y$, this corresponds to the necessary covariance of the dependency constraints of m in m''.
2. $\forall x, y \in \widetilde{m}, x \sqsubseteq_{m''} y \Rightarrow x \sqsubseteq_m y$, this will complete the demonstration of the property with its sufficient part.

Then the proof:

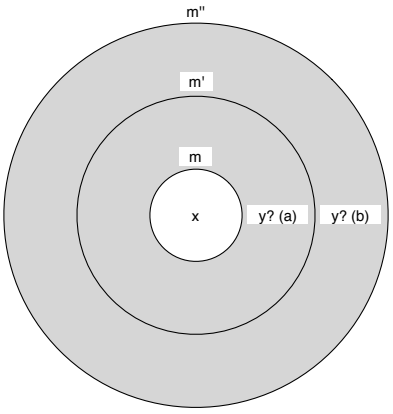

Fig. 13 $y \in \widetilde{m''} \setminus \widetilde{m}$.

1. Covariance necessity of the dependency constraints: $\forall x, y \in \widetilde{m}, x \sqsubseteq_m y \Rightarrow x \sqsubseteq_{m''} y$.
Indeed:
   - By assumption m is invariant in m', so m is covariant in m':
     - $\widetilde{m} \subseteq \widetilde{m'}$.
     - $(\forall x, y \in \widetilde{m}, x \sqsubseteq_m y \Leftrightarrow x \sqsubseteq_{m'} y) \Rightarrow (\forall x, y \in \widetilde{m}, x \sqsubseteq_m y \Rightarrow x \sqsubseteq_{m'} y)$.
   - By assumption m' is invariant in m'', so m' is covariant in m'':
     - $(\forall x, y \in \widetilde{m'}, x \sqsubseteq_{m'} y \Leftrightarrow x \sqsubseteq_{m''} y)$ $\Rightarrow (\forall x, y \in \widetilde{m'}, x \sqsubseteq_{m'} y \Rightarrow x \sqsubseteq_{m''} y)$.

  - Then the covariance: $\forall x, y \in \widetilde{m} \subseteq \widetilde{m'}$, $x \sqsubseteq_m y \Rightarrow x \sqsubseteq_{m'} y \Rightarrow x \sqsubseteq_{m''} y$.
2. And finally the remaining and sufficient condition for m to be invariant in m'': $\forall x, y \in \widetilde{m}, x \sqsubseteq_{m''} y \Rightarrow x \sqsubseteq_m y$.
   Indeed:
   - m being invariant in m' :
     - $\widetilde{m} \subseteq \widetilde{m'}$.
     - $(\forall x, y \in \widetilde{m}, x \sqsubseteq_m y \Leftrightarrow x \sqsubseteq_{m'} y)$ $\Rightarrow (\forall x, y \in \widetilde{m}, x \sqsubseteq_{m'} y \Rightarrow x \sqsubseteq_m y)$.
   - m' being invariant in m'':
     - $\widetilde{m'} \subseteq \widetilde{m''}$
     - $(\forall x, y \in \widetilde{m'}, x \sqsubseteq_{m''} y \Leftrightarrow x \sqsubseteq_{m'} y)$ $\Rightarrow (\forall x, y \in \widetilde{m'}, x \sqsubseteq_{m''} y \Rightarrow x \sqsubseteq_{m'} y)$.
   - Then the result: $\forall x, y \in \widetilde{m} \subseteq \widetilde{m'}, x \sqsubseteq_{m''} y \Rightarrow x \sqsubseteq_{m'} y \Rightarrow x \sqsubseteq_m y$.

□

Examples were given above as an introduction to this property. Other ones are shown in Figure 14 which also depicts some interesting covariance and invariance relationships that apply with regard to this property. Note that the presented submodels are possible extensions of the *HeatingSubCircuit* submodel (in Figure 11) when one wants to add regulation. The model $m4$ (bottom left) corresponds to the whole *Heating System*. $m3$ and $m3'$ are alternative submodels depending on whether *Regulators* or *Sensors* are envisaged first. $m4'$ is a combination of $m3$ and $m3'$. Some model elements of the overall system are not or incompletely taken into account in the submodels. For example:

- the class *Sensor* is missing in $m3$, the class *Regulator* is missing in $m3'$
- some of their features were ignored, for example *regulate* of *Regulator*, *notify* of *Sensor* or *connect* of *Sensor* (in $m3'$) .
- some ingredients were incompletely specified, for example *connect* of *Sensor* has no parameter in $m4'$ or the association *sensitives* in $m3'$ and $m4'$ has only one end.

Following is the formulation of models $m3$, $m3'$ and $m4'$ as $m = (\widetilde{m}, \sqsubseteq_m)$:

- $m3 = (\widetilde{m3}, \sqsubseteq_{m3})$
  - $m3$ ={*Boiler, temperature, float, on, off, pipes, Radiator, position, int, up, down, link, controls, Regulator, expectedT, setExpectedT*}
  - $\sqsubseteq_{m3}$:
    - *temperature* $\sqsubseteq_{m3}$ *float*
    - *temperature* $\sqsubseteq_{m3}$ *Boiler*
    - *on* $\sqsubseteq_{m3}$ *Boiler*
    - *off* $\sqsubseteq_{m3}$ *Boiler*
    - *pipes* $\sqsubseteq_{m3}$ *Boiler*
    - *pipes* $\sqsubseteq_{m3}$ *Radiator*
    - *position* $\sqsubseteq_{m3}$ *int*
    - *position* $\sqsubseteq_{m3}$ *Radiator*
    - *up* $\sqsubseteq_{m3}$ *Radiator*
    - *down* $\sqsubseteq_{m3}$ *Radiator*
    - *link* $\sqsubseteq_{m3}$ *Boiler*
    - *link* $\sqsubseteq_{m3}$ *Radiator*
    - *controls* $\sqsubseteq_{m3}$ *Radiator*
    - *controls* $\sqsubseteq_{m3}$ *Regulator*
    - *expectedT* $\sqsubseteq_{m3}$ *float*
    - *expectedT* $\sqsubseteq_{m3}$ *Regulator*
    - *setExpectedT* $\sqsubseteq_{m3}$ *float*
    - *setExpectedT* $\sqsubseteq_{m3}$ *Regulator*
- $m3' = (\widetilde{m3'}, \sqsubseteq_{m3'})$
  - $m3'$ ={*Boiler, temperature, float, on, off, pipes, Radiator, position, int, up, down, link, Sensor, ambientT, getT, sensitives*}
  - $\sqsubseteq_{m3'}$:
    - *temperature* $\sqsubseteq_{m3'}$ *float*
    - *temperature* $\sqsubseteq_{m3'}$ *Boiler*
    - *on* $\sqsubseteq_{m3'}$ *Boiler*
    - *off* $\sqsubseteq_{m3'}$ *Boiler*
    - *pipes* $\sqsubseteq_{m3'}$ *Boiler*
    - *pipes* $\sqsubseteq_{m3'}$ *Radiator*
    - *position* $\sqsubseteq_{m3'}$ *int*
    - *position* $\sqsubseteq_{m3'}$ *Radiator*
    - *up* $\sqsubseteq_{m3'}$ *Radiator*
    - *down* $\sqsubseteq_{m3'}$ *Radiator*
    - *link* $\sqsubseteq_{m3'}$ *Boiler*
    - *link* $\sqsubseteq_{m3'}$ *Radiator*
    - *ambientT* $\sqsubseteq_{m3'}$ *float*
    - *ambientT* $\sqsubseteq_{m3'}$ *Sensor*
    - *getT* $\sqsubseteq_{m3'}$ *float*
    - *getT* $\sqsubseteq_{m3'}$ *Sensor*
    - *sensitives* $\sqsubseteq_{m3'}$ *Sensor*
- $m4' = (\widetilde{m4'}, \sqsubseteq_{m4'})$
  - $m4'$ ={*Boiler, temperature, float, on, off, pipes, Radiator, position, int, up, down, link, controls, Regulator, expectedT, setExpectedT, Sensor, ambientT, getT, connect, sensitives*}
  - $\sqsubseteq_{m4'}$:
    - *temperature* $\sqsubseteq_{m4'}$ *float*
    - *temperature* $\sqsubseteq_{m4'}$ *Boiler*
    - *on* $\sqsubseteq_{m4'}$ *Boiler*
    - *off* $\sqsubseteq_{m4'}$ *Boiler*
    - *pipes* $\sqsubseteq_{m4'}$ *Boiler*
    - *pipes* $\sqsubseteq_{m4'}$ *Radiator*
    - *position* $\sqsubseteq_{m4'}$ *int*
    - *position* $\sqsubseteq_{m4'}$ *Radiator*
    - *up* $\sqsubseteq_{m4'}$ *Radiator*
    - *down* $\sqsubseteq_{m4'}$ *Radiator*
    - *link* $\sqsubseteq_{m4'}$ *Boiler*
    - *link* $\sqsubseteq_{m4'}$ *Radiator*
    - *controls* $\sqsubseteq_{m4'}$ *Radiator*
    - *controls* $\sqsubseteq_{m4'}$ *Regulator*
    - *expectedT* $\sqsubseteq_{m4'}$ *float*
    - *expectedT* $\sqsubseteq_{m4'}$ *Regulator*
    - *setExpectedT* $\sqsubseteq_{m4'}$ *float*
    - *setExpectedT* $\sqsubseteq_{m4'}$ *Regulator*
    - *ambientT* $\sqsubseteq_{m4'}$ *float*
    - *ambientT* $\sqsubseteq_{m4'}$ *Sensor*

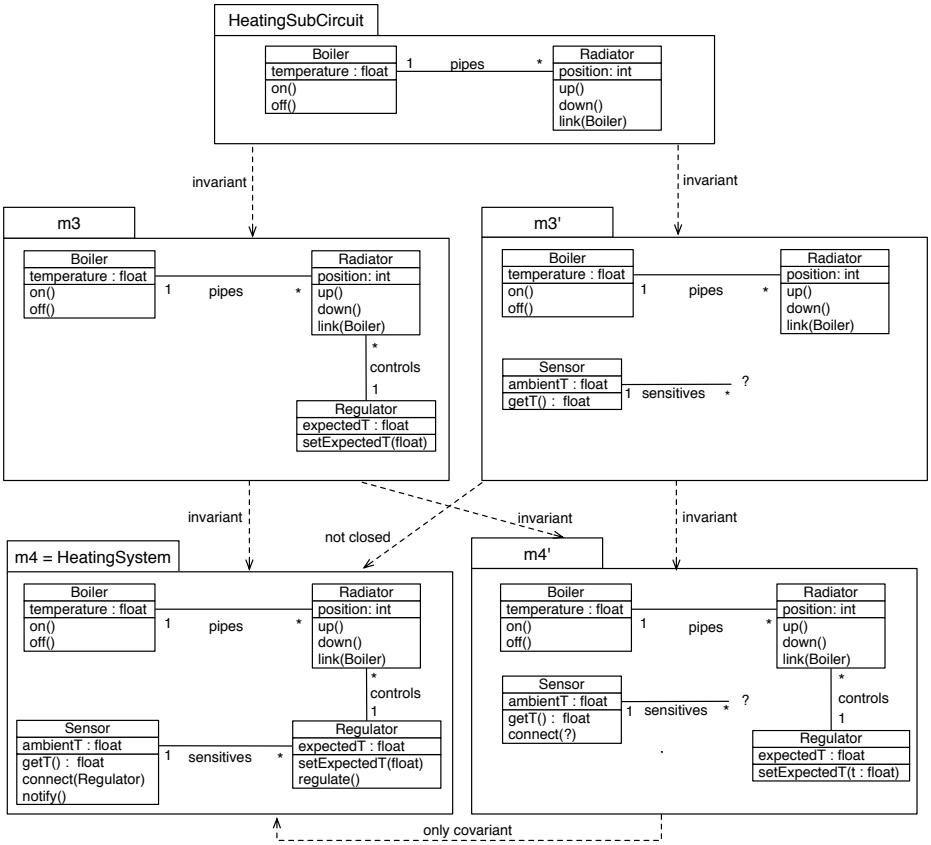

Fig. 14 Invariant Submodels

- $getT \sqsubseteq_{m4'} float$
- $getT \sqsubseteq_{m4'} Sensor$
- $connect \sqsubseteq_{m4'} Sensor$
- $sensitives \sqsubseteq_{m4'} Sensor$

Then examples of application of invariance transitivity:

- On the left side of the figure: $HeatingSubCircuit$ is invariant in $m3$ which is invariant in $HeatingSystem$, and indeed $HeatingSubCircuit$ is invariant in $HeatingSystem$.
- On the right side: $HeatingSubCircuit$ is invariant in $m3'$ which is invariant in $m4'$, and indeed $HeatingSubCircuit$ is invariant in $m4'$.

It is interesting to remark that submodels on the left side are well-formed ones contrary to those on the right and that invariance transitivity occurs in both cases. This example shows that submodel invariance and its transitivity property apply to any submodels, whether they are well-formed with regard to their metamodel or not. Indeed, the notion of invariant submodel is much more concerned with the conformance of submodels to surrounding ones, at the modeling level, than their conformance to the metamodel under consideration. It is worth noting that metamodeling constraints are only relevant as necessary ones in the present formalization as far as models of the same modeling level are concerned. This allows to consider any parts of a model, being well-formed or not, and then qualify their inclusion relationships. Note that this also applies to the previous notions of *closed* and *covariant* submodels notions and their properties.

Examine now "transverse" relationships in Figure 14:

- $m3$ is invariant in $m4'$, which confirms the fact that $HeatingSubCircuit$ as an invariant submodel of $m3$ is also invariant in $m4'$, as seen above.
- $m3'$ is not invariant, not even covariant, in $HeatingSystem$ because it is not closed in this model. Indeed *Regulator* is missing.
- $m4'$ is not invariant but only covariant in $HeatingSystem$. Indeed, $m4'$ is closed in $HeatingSystem$ contrarily to $m3'$ but it does not assert the following constraints imposed by $HeatingSystem$ on its model elements:
  - $sensitives \sqsubseteq_{HeatingSystem} Regulator$
  - $connect \sqsubseteq_{HeatingSystem} Regulator$
- And finally the fact that $m3'$ is not invariant in $HeatingSystem$ is consistent with $m4'$ not being invariant in this overall model, though $m3'$ was invariant in $m4'$.

5 Model Extraction

In the preceding sections an algebraic formalization of the submodel notion and its related inclusion relationship was grounded, aiming to clarify and characterize the phenomenon. Starting from a primitive interpretation of submodel inclusion based on their model elements, deeper submodels inclusion relationships which take into account their own structuring dimension were stated. Then it was shown that submodel transitivity could be guaranteed under some conditions, thanks to the notion of invariant submodel.

At an operational level, many model handling operators can profit from this formalization. As an example, we concentrate here on the operation which consists of producing a submodel of another one when selecting a subset of its model elements, that is *Model Extraction.*

With respect to the present formalization, *Model Extraction* from a model m due to the selection of a subset s of its model elements can be defined as the following $model_m(s)$ operator. It consists in associating to s the dependency constraints of m that apply to s.

Definition 8 (Model Extraction Operator)
Let a model $m = (\widetilde{m}, \sqsubseteq_m)$, for any subset s of its model elements ($s \subseteq \widetilde{m}$), the operator $model_m(s)$ allows to extract the submodel $m' = (s, \sqsubseteq_{m'})$ of m such as:
$\forall x, y \in s, x \sqsubseteq_m y \Rightarrow x \sqsubseteq_{m'} y$,
that is $Gr(model_m(s)) = Gr(m)/s$.

For example, here are two samples of model extraction from the *HeatingSystem* class model (see Figure 4) of the paper:

- Selecting the following subset of model elements $s =$ {*temperature, float, Boiler, on, off, pipes, Radiator, position, int, up, down, link*} allows to extract the model *HeatingSubCircuit* of the heating sub-circuit (already used in many preceding examples), that is: $model_{HeatingSystem}(s) = HeatingSubCircuit$.
- Selecting this other subset of model elements $s =$ {*Regulator, expectedT, setExpectedT, float, regulate, sensitives , Sensor, ambientT, getT, connect, notify*} extracts the model of the regulation sub-circuit (see Figure 15), that is:
  $model_{HeatingSystem}(s) = RegulationSubCircuit$.

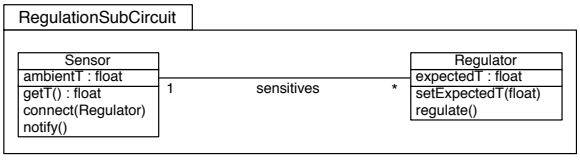

Fig. 15 Regulation Circuit Submodel

This operator has interesting properties that will be studied in the following. For this purpose, let us insist on the fact that it involves two parameters :

- s, the selected subset of model elements of the model m under consideration.
- m, this model itself.

So that it will be convenient to examine what happens when it is s or m which alternatively varies, in particular when they respectively "grow", that is:

- Considering a superset s' of s, how do the resulting models $model_m(s)$ and $model_m(s')$ relate? It will be shown (Property 7) that when s and s' are closed subsets of m, the model produced by $model_m(s)$ is guaranteed to be an invariant submodel of $model_m(s')$.
- Considering any super-model m' of m, how do the resulting models $model_m(s)$ and $model_{m'}(s)$ relate? It will be shown (Property 9) that, when m is an invariant submodel of m', s being a closed subset of m, the model produced by $model_m(s)$ is the same as the one produced by $model_{m'}(s)$.

But, first and foremost, primitive properties of the operator may be set out. Given that $s \subseteq \widetilde{m}$ in the definition of the $model_m(s)$ operator, two cases claim to be considered:

1. The special case where $s = \widetilde{m}$.
2. The general case : $s \subsetneq \widetilde{m}$.

First case ($s = \widetilde{m}$) corresponds to the situation where s covers the whole set of model elements of the model under consideration. As expected, the resulting model is this whole model itself.

Property 4 (Model Extraction applied to the whole set of model elements of a model)
Let some model m,
$model_m(\widetilde{m}) = m$.

Proof :
$model_m(\widetilde{m}) = (\widetilde{m}, Gr(m)/\widetilde{m}) = (\widetilde{m}, Gr(m)) = m$.
□

This property is in fact related to the more general following one which characterizes the *idempotence* of the $model_m(s)$ operator with regard to the involved model m. Indeed, when one tries to extract, multiple times, the model corresponding to s from the model produced by the operator itself, the resulting one does remain the same.

Property 5 (Model Extraction Idempotence)
Let a model m, for any subset s of its model elements ($s \subseteq \widetilde{m}$),
$model_{model_m(s)}(s) = model_m(s)$.

Proof :
Let $m' = model_m(s)$,
$model_{model_m(s)}(s) = model_m(s) \Leftrightarrow model_{m'}(s) = m'$.
By definition, $\widetilde{m'} = \widetilde{model_m(s)} = s$, so that:
$model_{m'}(s) = m' \Leftrightarrow model_{m'}(\widetilde{m'}) = m'$,
which is true, thanks to Property 4.

□

Second, examine the more general case: $s \subsetneq \widetilde{m}$. The following property (Property 6) states that, when s is a *closed* subset of model elements of m, the corresponding extracted model is guaranteed to be an *invariant submodel* of m.

Property 6 (Model Extraction applied to a Closed Subset)
Let a model m, for any subset s of its model elements ($s \subseteq \widetilde{m}$),
s closed in m
⇒
$model_m(s)$ is an invariant submodel of m.

Proof :
$model_m(s)$ is an invariant submodel of m iff:

- $\widetilde{model_m}(s)$ is closed in m. By definition $\widetilde{model_m}(s) = s$ which is closed in m, by assumption.
- $Gr(m)/\widetilde{model_m}(s) = Gr(model_m(s))$. Indeed, by definition of the $model_m$ operator:
  - $\widetilde{model_m}(s) = s$
  - $Gr(model_m(s)) = Gr(m)/s$.

□

For example, consider Figure 12 and $s = \{$*Boiler, temperature, float, on, off, Radiator, up, down*$\}$ of model elements of $HeatingSubCircuit$, it is easy to verify that the model produced by $model_{HeatingSubCircuit}(s)$ is $m2$ which is indeed invariant in $HeatingSubCircuit$. In addition, this example is an opportunity to show the selection applied by the operator. Indeed, s is the set of model elements of $m2$, but recall that it is also the one of $m2'$ (as pointed in Section 4.2) and $m1$ (see Section 4.3). Then it is interesting to observe that, among these submodels (*m1, m2, m2'*) which share the same subset s of model elements of $HeatingSubCircuit$, the result of the extraction is $m2$, which is the only invariant submodel of this model, so stable, of the three.

It is worth noting that in both cases of this primary analysis, s appears to be a closed subset of model elements of m, either because s was the entire $\widetilde{m}$ set (Property 4), or by assumption (Property 6). The next case would be to consider any subset $s \subsetneq \widetilde{m}$ not even closed in $\widetilde{m}$. Though it is easy to be convinced that this situation has no or little relevance. The subset s not being closed in m, there is no chance for the submodel produced by $model_m(s)$ to be interestingly qualified as *closed* nor *covariant*, even less *invariant* in the model m under consideration. Indeed these attributions necessarily depend, by definition, on s being at least closed in m. In other words s being unspecified, so is the model produced by $model_m(s)$.

For example, consider again the model *HeatingSubCircuit* and now its subset of model elements $s = \{$*temperature, float*$\}$ then $model_{HeatingSubCircuit}(s) = m0$ (Figure 12), s being not closed in $HeatingSubCircuit$:
$closure_{\sqsubseteq_{HeatingSubCircuit}}(\{temperature, float\})$
$= \{temperature, float, Boiler\}$
$\neq \{temperature, float\})$,
so is $m0$.

After this preliminary on primitive properties of the $model_m(s)$ operator, let us now examine, as announced, what happens when the subset s or the model m under consideration grows.

First, consider a superset s' of s, both being subsets of model elements of a model m under consideration ($s \subseteq s' \subseteq \widetilde{m}$). Then it appears that, when s and s' are closed subsets of $\widetilde{m}$, the extracted model produced by $model_m(s)$ is an invariant submodel of the one produced by $model_m(s')$.

Property 7 (Model Extraction applied to included subsets)
Let a model m and two subsets s, s' of its model elements such as : $s \subseteq s' \subseteq \widetilde{m}$,
s and s' are closed in m
⇒
$model_m(s)$ is an invariant submodel of $model_m(s')$.

Proof :
Let us note:

- $S = model_m(s) = (s, Gr(m)/s)$
- $S' = model_m(s') = (s', Gr(m)/s')$

S is an invariant submodel of S' iff:

1. S is closed in S'.
2. The dependency constraints of S are invariant in S'.

Proof of 2.
That is to show: $Gr(S')/s = Gr(S)$. Indeed:
$Gr(S')/s = (Gr(m)/s')/s$
$s \subseteq s' \Rightarrow (Gr(m)/s')/s = Gr(m)/s$
$Gr(m)/s = Gr(S)$, so the result.

Proof of 1.
That is to show: $closure_{\sqsubseteq_{S'}}(\widetilde{S}) = closure_{\sqsubseteq_{S'}}(s) = s$.
Indeed, following the preceding proof:
$Gr(S')/s = Gr(m)/s$
so that: $closure_{\sqsubseteq_{S'}}(s) = closure_{\sqsubseteq_m}(s)$.
Since s is closed in m, we have: $closure_{\sqsubseteq_m}(s) = s$,
so the result
$closure_{\sqsubseteq_{S'}}(\widetilde{S}) = closure_{\sqsubseteq_{S'}}(s) = closure_m(s) = s$.

□

For example, consider Figure 16[3] and $s = \{$*Boiler, Radiator, pipes, Sensor*$\}$ (that is $\widetilde{m2}$), $s' = \{$*Boiler, Radiator, pipes, Sensor, sensitives*$\}$ ($= \widetilde{m3'}$), $m4'$ being the source model under consideration, we have :

[3] An updated version of Figure 14 with same $m3'$, $m4'$ and $m4$ submodels but new $m1$, $m2$ and $m3$ submodels. Attributes were omitted to simplify the figure.

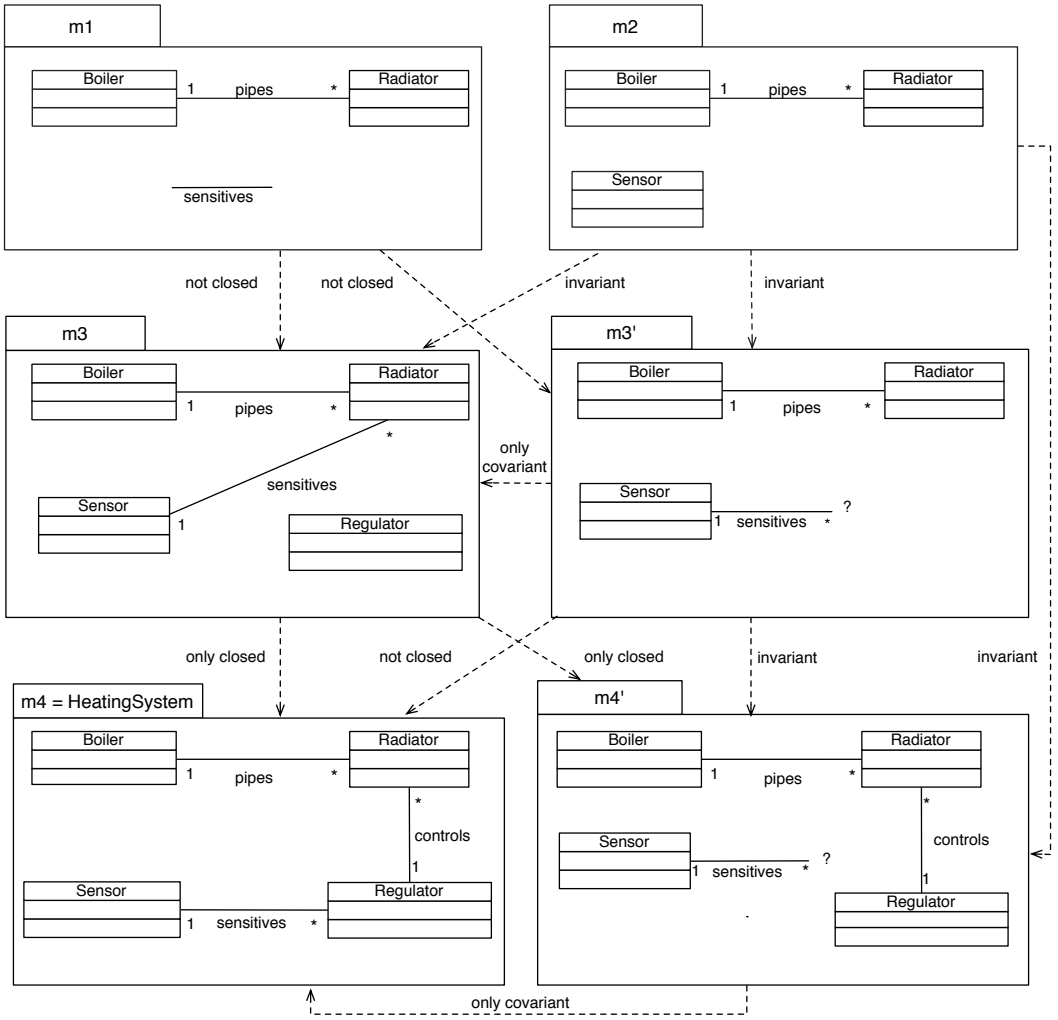

Fig. 16 Samples for Property 7 and following.

- $s \subseteq s' \subseteq \widetilde{m4'}$
- s and s' closed in $m4'$
- $model_{m4'}(s) = m2, model_{m4'}(s') = m3'$
- and indeed $m2$ is invariant in $m3'$.

As a counter-example, in the same figure, consider now $s = \widetilde{m3'}$, $s' = \{$*Boiler, Radiator, pipes, Sensor, Regulator, sensitives, controls*$\}$ (that is $\widetilde{m4'}$), and the model $m4$ under consideration, we have :

- $s \subseteq s' \subseteq \widetilde{m4}$
- s' closed in $m4$
- but s, being not closed in $m4$, $model_{m4}(s) = m3'$, is not invariant (not closed) in $model_{m4}(s') = m4$.

Second, what happens when it is the model m under consideration itself which is growing? This will be established in the final Property 9.

Before that, let us state an intermediate property which is a particular case of Property 6 when the closed subset under consideration issues from an invariant submodel of the involved model. It is an expected property of the $model_m(s)$ operator that well characterizes its consistency with the notion of invariant submodel. This property relies on the preceding ones, that is why it is positioned at this stage of the study.

Property 8 (Model Extraction Applied to a Closed Subset Issued from an Invariant Submodel)
Let two models m, m'
m is invariant in m'
$\Rightarrow$
$model_{m'}(\widetilde{m}) = m$

Proof : $model_{m'}(\widetilde{m}) = m$ iff:

1. $\widetilde{model_{m'}}(\widetilde{m}) = \widetilde{m}$
2. $Gr(model_{m'}(\widetilde{m})) = Gr(m)$

Proof of 1.
By construction: $\widetilde{model_{m'}}(\widetilde{m}) = \widetilde{m}$

Proof of 2.
m being invariant in m' (by assumption), we have $\widetilde{m} \subseteq \widetilde{m'}$ so that Property 7 applies:

- $\widetilde{m} \subseteq \widetilde{m'}$
- $\widetilde{m}$ closed in m' (m being invariant in m')
- $\widetilde{m'}$ closed in m'
- $\Rightarrow model_{m'}(\widetilde{m})$ is invariant in $model_{m'}(\widetilde{m'})$

That is (thanks to Property 4, $model_{m'}(\widetilde{m'}) = m'$)
$model_{m'}(\widetilde{m})$ is invariant in m' and then:
$Gr(model_{m'}(\widetilde{m})) = Gr(m')/model_{m'}(\widetilde{m}) = Gr(m')/\widetilde{m}$.
But, m being invariant in m', $Gr(m')/\widetilde{m} = Gr(m)$ so the result: $Gr(model_{m'}(\widetilde{m})) = Gr(m)$.

□

Finally, the following property states that, for any subset s of model elements of m, the model produced by $model_m(s)$ will not vary when one considers any supermodel m' whose m is an invariant submodel, that is $model_{m'}(s) = model_m(s)$. More formally:

Property 9 (Model Extraction from Included Invariant Submodels)
Let two models m, m', for any subset s of model elements of m:
m is invariant in $m' \Rightarrow model_{m'}(s) = model_m(s)$.

Proof :
By definition:

- $model_m(s) = (s, Gr(m)/s)$
- $model_{m'}(s) = (s, Gr(m')/s)$

The two models have the same set of model elements (s). Let us show, now, that they have also the same dependency constraints, that is: $Gr(m)/s = Gr(m')/s$.
m being invariant in m', $Gr(m')/\widetilde{m} = Gr(m)$, so that $Gr(m)/s = Gr(m')/\widetilde{m}/s$. Since $s \subseteq \widetilde{m}$, we have $Gr(m')/\widetilde{m}/s = Gr(m')/s$, then the result.

□

For example, consider again Figure 16 with $s = \{$*Boiler, Radiator, pipes, Sensor*$\}$ (that is $\widetilde{m2}$), $m = m3'$ and $m' = m4'$, we have:

- $s \subseteq \widetilde{m3'}$
- $m3'$ invariant in $m4'$
- $model_{m3'}(s) = m2$
- $model_{m4'}(s) = m2$

So the equality of extracted models from invariant submodels.

When the submodels under consideration are only closed or covariant (not invariant), the property is not ensured. Here are counter-examples in each case.

Consider $m = m3$, $m' = m4$, $m3$ being only a closed submodel of $m4$, and $s = \{$*Boiler, Radiator, pipes, Sensor, sensitives*$\}$ ($= \widetilde{m3'}$). Here we have :

- $s \subseteq \widetilde{m3}$
- $m3$ only closed in $m4$
- $model_{m3}(s) = (s, Gr(m3)/s))$
- $model_{m4}(s) = (s, Gr(m4)/s))$
- *Gr(m3)/s=*{*(pipes,Radiator), (pipes,Boiler), (sensitives,Sensor), (sensitives,Radiator)*}
- *Gr(m4)/s=*{*(pipes,Radiator), (pipes,Boiler), (sensitives,Sensor)*}

Therefore, due to distinct sets of dependency constraints, extracted submodels are not equal.

A covariant counter-example is $s = \{$*Boiler, Radiator, pipes, sensitives*$\}$ (that is $\widetilde{m1}$), $m = m3'$ and $m' = m3$. Then:

- $s \subseteq \widetilde{m3'}$
- $m3'$ only covariant in $m3$
- $model_{m3'}(s) = (s, Gr(m3')/s) = m1$
- $model_{m3}(s) = (s, Gr(m3)/s)$
- *Gr(m3')/s=*{*(pipes,Radiator), (pipes,Boiler)*}
- *Gr(m3)/s=*{*(pipes,Radiator), (pipes,Boiler), (sensitives,Radiator)*}

The extracted submodels get different structuring constraints and so are not equal.

6 Application: a Submodel Engine for EMF Models

This section presents an application of the previous results to build an extensible submodel engine integrated with Eclipse and its modeling framework (EMF). It also discusses the customization of this engine for specific kinds of model and illustrates its use through the presentation of a tool designed for visualizing the relation graph of a set of models.

The engine provides a set of core functionalities to determine how models relate according to the submodel relationships, ranging from simple inclusion of elements sets to invariant submodels. More precisely, the provided functionalities enable:

- to determine if an input model includes all the elements of a second input model;
- to determine if an input model is a closed submodel of a second input model;
- to determine if an input model is a covariant submodel of a second input model;
- to determine if an input model is an invariant submodel of a second input model;
- to get the strongest relationship between two models if it exists;
- to compute the closure of a model with respect to a second one.

All these functionalities are available as a service offered by the engine to other modeling tools included in the Eclipse environment. Tools can exploit the results returned by the service for many modeling tasks like comparing a selected set of models, retrieving relevant models from libraries using the various submodel notions or organizing several models in a network of relationships.

One key feature of the engine is that the previous functionalities are offered for any kind of EMF models. Indeed, our engine is extensible and can be easily customized to work with multiple metamodels given that models conforming to a metamodel can be transformed in the formulation retained in this paper, that is as a set of model elements and a partial order derived from their dependency constraints.

Figure 17 shows the architecture of the extensible engine which follows the plugin-based style promoted by the Eclipse environment. This architecture is composed of two component types: the core component and one or several EMF model adapters. The core component provides the functionalities mentioned above and made them available as a service to the rest of the platform. This service is automatically recorded in the registry of all services available in the platform. EMF Model adapters are components that extend the engine for specific metamodels. They are recorded in the plugin registry and are discovered automatically by the core component.

The core expects that EMF model adapters conform to an interface to execute their functionalities. Through this interface, the core component queries a model adapter to decide whether it is suitable for an input EMF model and, if so, asks this model adapter to convert the input EMF model into the representation handled by the core. Optionally, an EMF model adapter can also customize how elements of two input EMF models must be matched during operations performed by the core[4]. At the implementation level, the core and model adapters component are designed as Eclipse plugins which are automatically connected thanks to the mechanisms of plugin extension point and plugin contribution offered by the platform.

Currently, we have developed and experimented three EMF model adapters for the engine. One of them extends the engine for Ecore models (Eclipse implementation of the EMOF specification). This model adapter enables the exploration of submodels within the space of metamodels. The two other adapters target UML2 models. A first one is dedicated to UML class models. By example, all the samples given in the paper including those which refer to ill-formed models like $m3'$ and $m4'$ of Figure 14 have been verified thanks to this adaptation. The second one is related to UML2 collaboration models and gives the capacity to compare such models. Elements of collaboration models that are taken into account for submodel notions are roles, role types, connectors between roles and collaboration uses. Figure 18[5] gives an example illustrating the application of our engine to four collaboration models in the area of telephony services. These collaborations represent partial or complete specification of ingredients involved in such services. *RegisterUse* collaboration captures the use of an existing collaboration dedicated to registration of user adresses. *ServerBasedRegistration* refines the previous collaboration to express registration of user addresses with a server. Note that these two examples are not well-formed, as they contains a *CollaborationUse* element without its complete set of role bindings. *UserCall* and *RedirectedUserCall* are well-formed collaboration models expressing two variants of user calls, respectively direct ones and indirect ones relying on an intermediate server for callee registration and localization. By using our engine, we can compare such models and get the relationships shown in Figure 18, resulting in a better comprehension of how these models relate regarding their respective parts.

As indicated previously, the functionalities offered by the engine and its extensions can be exploited to construct powerful modeling tools that require submodel notions similar to the ones presented in this paper. Kinds of tool that can profit from such functionalities are for instance content-based querying, submodel browser, submodel validator, transformation engine and model editor with automatic submodel completion. Hereafter, we give an example of a visualization tool that was developed on top of our engine to get a graphical view of submodel relationships existing within a base of models or model fragments. Figure 19 presents a snapshot of this tool for a subset of the class models used in this paper. The content of the presented view is obtained by using the engine functionality for computing the strongest relationship between selected models, each kind of resulting relationship being represented by a distinct color. Such a view can be obtained for other kinds of EMF model (like collaboration models) if a model adapter exists.

As shown in the figure, the tool organizes all the selected models in a network of relationships that are computed thanks to the engine and displays this network graphically. Among other features, the tool allows to filter the network on a specific relationship and allows to reduce the focus on a particular model while navigating from one model to another inside the network. In addition, the tool gives the capacity to increase or reduce the set of selected models so that they can be compared dynamically. Such a visualization tool can aid exploration, discovery and retrieval of submodels. It can also support questions about submodel compatibility and help to detect potential common parts and find derivation link between existing models.

More generally, we believe that these tools can be useful to organize (or megamodel) big repositories of models. Currently, we are working on a version of this tool that can connect to remote model repositories like EMF Store and Eclipse CDO Model Repository.

[4] Name matching of elements is an example used in the present application.

[5] For reasons of space, only the formulation of the *UserCall* Collaboration is detailed.

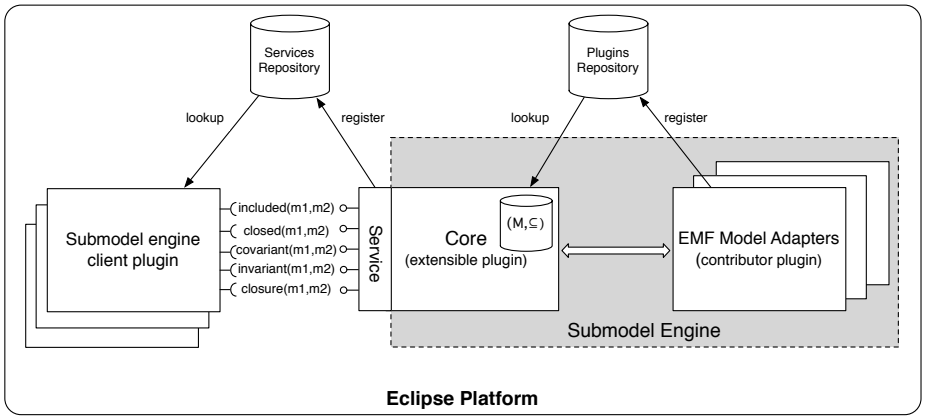

Fig. 17 Engine Architecture

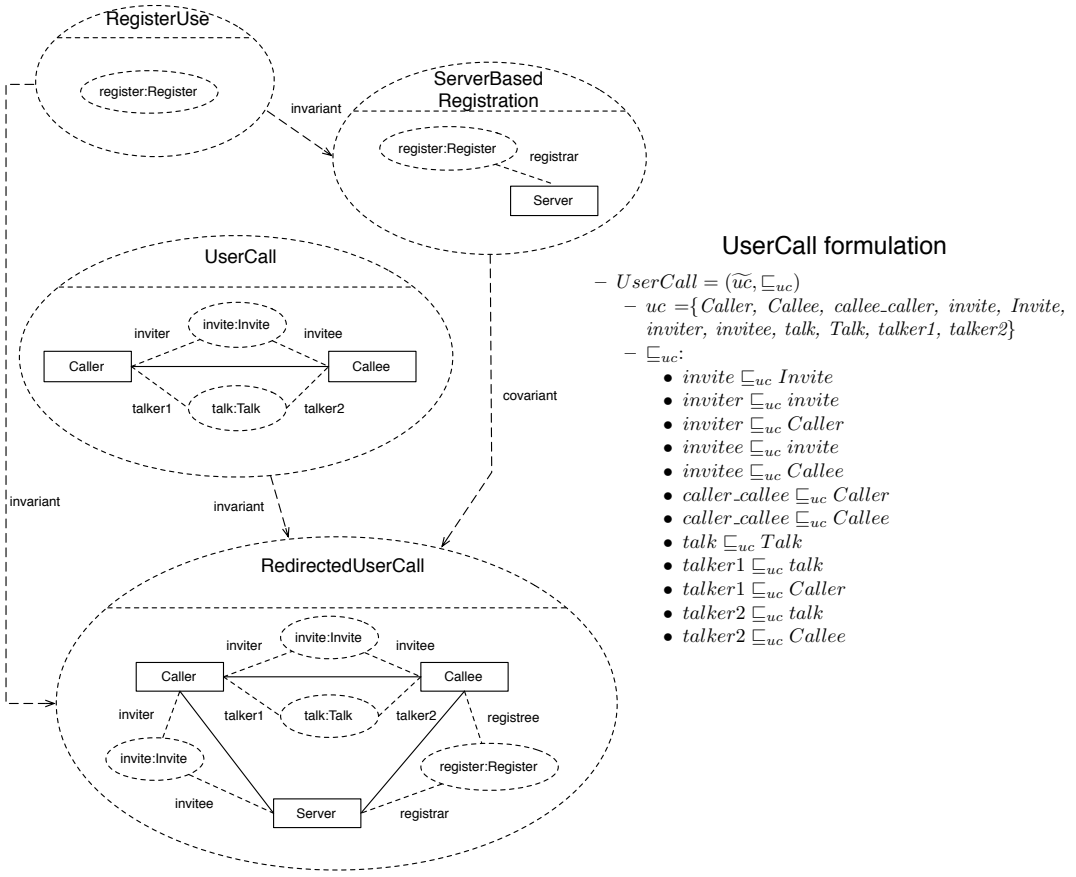

Fig. 18 Collaboration submodels

7 Related Works

In this work, we set a formalized notion of submodel and its corresponding relationship which completes the set of relationships, such as conformance and representation ones. This contributes to the structuring of the model space [27,8], and allows better understanding and comparison of involved artifacts. In addition, model extraction was analyzed on the basis of this formalization. This constitutes a step to improve modeling process and modeling tools. Although submodels are very useful in MDE practices and underly many modeling tasks, this notion has not been widely studied, leading to loose definitions and implicit meaning. As indicated in [42], the lack of clear definition is prone to misunderstanding and prevents for developing systematic submodel-based methods and tools.

In [26], the authors also propose a formalized notion of submodel that is general like our proposal. The main

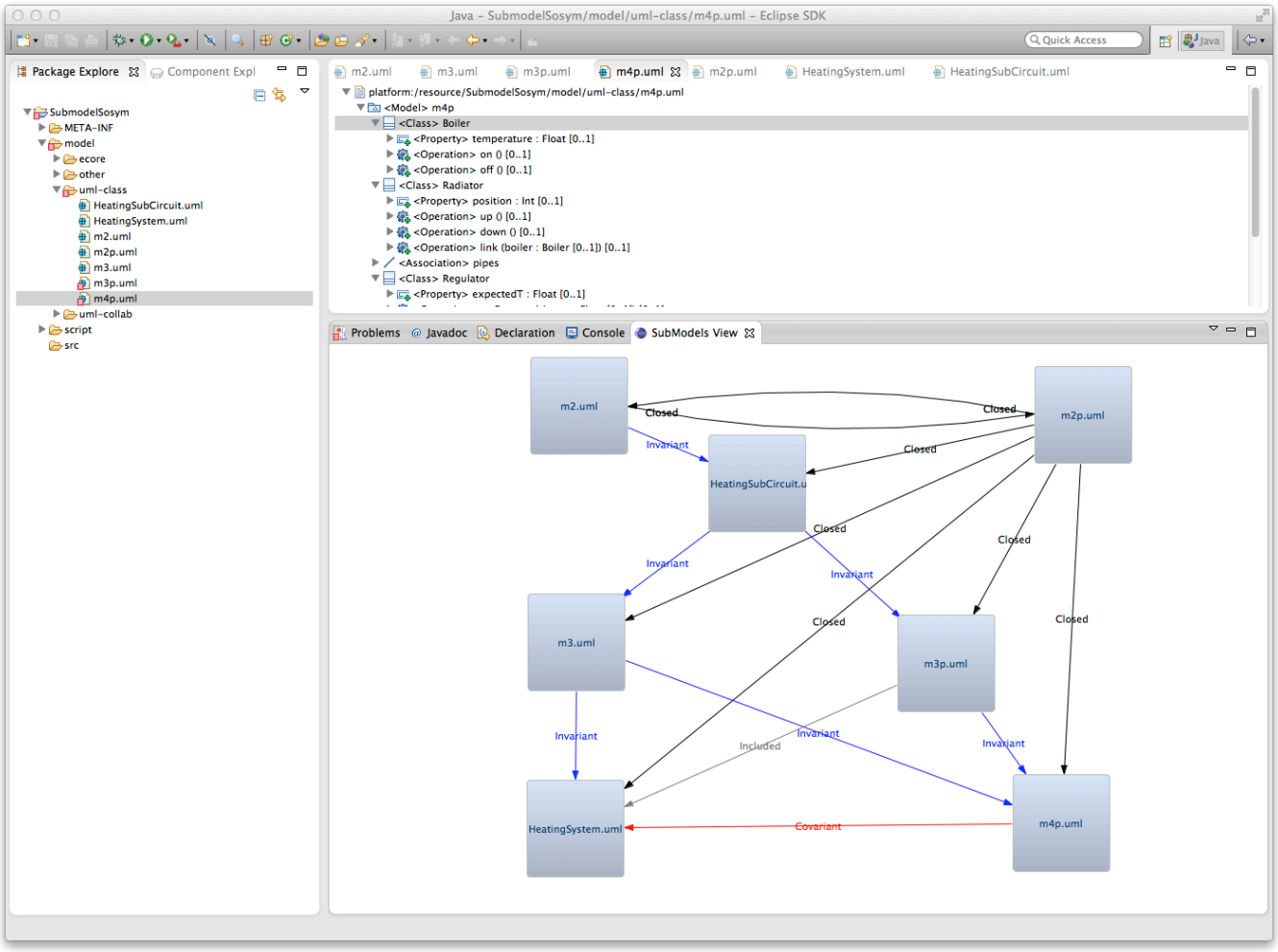

Fig. 19 Tool for visualizing submodel relationships

concern for the submodel notion proposed in this work is better model comprehension through decomposition of a model into submodels that conform to the same metamodel. Here, the conformance of submodels is ensured by the inclusion of elements between a model and its submodels and several conditions related to the metamodel such as type compatibility and multiplicity conditions. The motivation for this kind of submodel notion is the ability to view and manipulate the submodels using the same tools as the original model. From this submodel notion, the authors propose an algorithm to decompose a model into a submodel lattice. This lattice is constructed in a straightforward manner by relying on the identification of fragmentable links in the metamodel. This work and ours are complementary but differ in two main respects. First, they focus on the conformance of submodels with respect to the metamodel while we focus on the structural constraints between models at the same level. Second, their goal is mainly to automatically build a lattice of relevant submodels from a source model. The aim of our work is more general for the provision of a characterization of the submodel notion and related model operators, such as model extraction.

In the following, we present a synthesis on areas which make directly use or resort to submodels.

Submodels for Model Operation Efficiency

In [42], the authors propose to use model fragments for the purpose of efficiency in performing model operations. In this context, a model fragment is an independently mutable and distinct entity which is a copy of a piece of a model. Such an entity consists of instances of metaclasses, meta-associations, and meta-attributes that are (possibly incomplete) copies of instances in the source model. According to their definition, a model fragment can be modified, and there is no requirement to propagate the changes to the source model or vice versa. The authors also present a model fragment selection mechanism that aims to determine parts of models containing relevant information from the viewpoint of an operation. This mechanism consists of two consecutive steps called

implication and *projection*. In the implication step, a set of model elements is selected from an input model, thanks to rules expressed in OCL. In the projection step, the set of preceding elements is projected onto a pruned version of the input metamodel to constitute the final model fragment[6]. A comparison between this work and ours shows some main differences. Due to a different focus, the notion of model fragment given in this work is distinct from ours and is not defined in algebraic way but as the result of a specific extraction process. Contrarily to our work, the authors do not formalize their notion of model fragment and their extraction mechanism. This work is also less general than ours in the sense that it restricts its attention to the UML language and the context of model operation. Another difference is the explicit projection of the resulting model fragment to a pruned metamodel which is not a concern in our work. Finally, both approaches also differ in the way they address consistency of model fragments: in their approach, this consistency is ensured manually by the designer through OCL-based rules and a given pruned metamodel while our approach for consistency is systematic and relies on algebraic rules which apply to sets of model elements and their structural constraints.

Submodels and Model Compatibility

Model Compatibility is concerned with model substitutability in operation and tools. This currently calls for metamodel inclusion. While we focus here only on model, our submodel notion can also be applied at the metamodeling level to characterize model substitutability.

Model Typing and *Subtyping* presented in [47,22] state strategies for model substitutability through metamodel inclusion. A metamodel is considered as a collection of interconnected objects. Substitutability is defined by a matching relationship between two metamodels which is based on the preservation of classes and the related dependent classes contained in the first metamodel into the second one. According to this matching relationship, a metamodel that has fewer corresponding classes, properties or relationships than another metamodel can be stated as a supertype of the latter [41]. We also see a proximity between our work and *Model Inheritance*, a concept that has currently received little attention in existing works except in [30]. In this work, the author explores inheritance between model types with an emphasis on compatibility considerations, i.e backward and forward compatibility for tools. He presents several forms of inheritance according to the inclusion of model type intension (metamodel and its constraints) and model type extension (its model instances). Some similitaries exist between our variants of submodel notion and forms of model inheritance relying only on model intension like *compatible pruning* and *conceptual containment*.

More generally, as mentioned in [22], Model Typing is a major issue in the MDE which has not been sufficiently studied. All these works, including ours, contribute to this challenge and lay basis for further investigation and comparison.

Submodel Extraction

We found works that have proposed operators for extracting a part of a model. Most of these works take place in the field of model management and are motivated by the providing of a rich set of operators for manipulating and integrating models and metamodels. The work presented in [35] offers a set of high-level algebraic operators that apply to any type of models or model management applications. One of the generic operator named *Extract* is dedicated to the extraction of a model fragment. This operator requires an input model and a set of selected elements. It returns a well-formed part of the input model satisfying some properties plus a mapping that relates them. A *Delete* operator, defined as extraction of an unselected portion of a model, is also proposed. Paper [39] is in the same vein and proposes *mega operations* to specify integration of models covering distinct concerns or domains at the meta-level and derive the actual integration at the model level. Weaving and sewing mega-operations are defined using a set of primitive operators where one is dedicated to the pruning of a metamodel, a kind of extraction based on the riding of all unnecessary meta-model elements in a meta-model. In this study, we only focused on model extraction but deeply analyze its properties thanks to our submodel groundings. The same could be applied to operators presented in these works, notably to study how outputs vary depending the variation of input sets and models.

Template-Based Modeling

The current submodel notion originates from our works on views and their reuse through templates [10,12,11, 36]. Templates are models that expose some of their elements as parameters and allow to produce other models through parameter substitutions. Through parameterization, templates provide a form of generic modeling that allows to capture model of recurrent structures so that they can be reused in multiple modeling contexts. This concept is used in many areas aiming to improve model reuse and appears as a cornerstone of model capitalization pushed by the MDE. Applications of model templates are modeling of generic classes (such as C++ templates or generics in Java), pattern formulation [49, 4], modeling of reusable subjects [14], Catalysis frameworks [16] or AOM (Aspect Oriented Modeling) [36,20].

[6] In other words, all instances of unneeded meta-classes, meta-attributes or meta-associations are removed from the model fragment.

In UML2, model template exists as a standard construct associated to a binding mechanism and is general enough to render most of the applications cited previously, depending on the status of the parameter set. In particular, when this parameter set forms a submodel, we obtain model templates which add ingredients to the submodel formed by the parameters. In our previous work [36], we deeply studied the composition of such templates, the correctness of composition chains and their alternative ordering capacities. All these features rely heavily on fully structured and consistent submodels. Thanks to the present work, formalization grounded on the submodel notion is achievable. Such formalization will help to better characterize the properties and usages of these templates. It will also enable to investigate formally degrees of type-safety for parameters and their subtyping, potential issues with binding and the conditions under which template instantiation yields conformance. We are currently working on this topic.

Aspect-Oriented Modeling

In line with the preceding works, Aspect-Oriented Modeling raises the idea of separation of concerns, so submodels, in the design of software models. Over the last years, many AOM techniques have been proposed [17, 38,50,37,28]. All these techniques provide a notion of model-based aspect and a model weaving process that produces new models by injecting ingredients from such aspect into relevant elements of input base models. Model-Based Aspects are typically made of pointcuts and advices. Pointcuts are abstract model elements defining where to affect the base model and the advices are corresponding elements defining how to extend those identified by the pointcuts. In most of the AOM approaches, pointcuts and advices are expressed by two related models with corresponding elements and the model of pointcuts is a submodel of the model of advices [50,28,31]. However, this relationship between these two models is most of the time either assumed implicitly or loosely defined. This has the consequence that the consistency between the two models is not handled or is only ensured partially. There are several ways to alleviate this problem but the notion of submodel presented in this paper can be a convenient approach in order to get a rigorous definition of the relationship between the models forming an aspect and a way to elaborate their consistency.

Model Slicing

The idea behind model slicing comes from the application of program slicing to the domain of models. It is a technique that consists in extracting a part of a model called a *slice* for a specific purpose. Model slicing is mainly exploited to assist and ease model comprehension when building huge models. Works have been investigated in the context of UML models [29,25,32], feature models [2] and also for any metamodel [9]. Mechanisms for selecting elements in order to form a slice are more or less sophisticated. In the simplest form, selected elements are given by designating and enumerating them in a set. Others complex forms such as ones based on a dedicated model [9] or predicates over models [25] exist. Among these approaches, there are some variations about the nature of produced slices and it is not always mandatory that the set of elements constituting the slice forms a model [23]. Generally, approaches that aim to construct submodels for slices only consider their conformance with respect to the metamodel and ignore structural constraints between elements themselves. As presented in this paper, such constraints are also important for obtaining consistent submodels and we think that the present work could be useful to improve the consistency of submodels extracted by such approaches. Furthermore, another interest of our work lies in our formalized notion of submodel that can help to characterize the kind of model fragment produced by different model slicing approaches.

Metamodel Pruning

In the same vein, some works have investigated the possibility of extracting some parts of metamodels to make them more understandable and more manageable during transformation and evolution. Paper [5] addresses the problem of decomposing the UML metamodel into smaller metamodels corresponding to the diagram types of UML. Metamodel pruning [40] is a generic approach that outputs an effective subset metamodel of a possible large input metamodel, such as whole UML one. In that work, the output metamodel is produced by removing elements from the input metamodel while preserving a set of selected types and properties as well as all their dependencies. The resulting metamodel is intended to be a supertype of the original metamodel in the sense of [47]. The work described in [43] addresses the complexity of the current UML 2.0 metamodel and proposes the general idea of views for metamodels in order to solve the problem. A view consists of a subset of all classes and relationships directly or indirectly related to one or several concepts in the metamodel. The authors propose to develop tools capable of querying a metamodel and automatically derive specified views from the queries. Query and extraction capabilities of such tools are illustrated for the UML metamodel through views corresponding to the different diagram types. In all these works, we can observe that there is a need to extract a part of a metamodel. Given that a metamodel is a model, it is possible to envision the application of submodels presented here for this area. Although we only presented illustrations

of our proposal applied to models, the resulting notions and properties are not restricted to this modeling level and are sufficiently general to be applied for extracting a consistent "sub-metamodel". However, while our work can serve as an interesting basis for extracting valid sub-metamodels, we consider that further work is necessary before fully exploiting the results at the metamodeling level. In particular, the relationship between our submodel notion applied to the instance model level and the same applied to the metamodel one is an interesting issue that we let for future works.

Submodels and Model Transformation

We also foresee applications of our work in the area of model transformation. Most often, a model transformation needs to manipulate some specific parts of a model instead of the whole model. For instance, the designers may want to transform only the behavioral parts of a design. Generally, the use of model parts is not assumed in main existing works and engines like QVT [21] or ATL [24] and a transformation only operates on the whole model. Few works such as [41,18] explicitly use model parts in model transformation but they generally rely on ad-hoc definitions. Though, by enabling partial models, transformations could benefit of qualities like locality, for example to reduce the impact of model evolution on transformation maintenance or limit the responsibilities of specific transformation in case of erroneous results. Other benefits of transformation restricted to a portion of a model are performance and composability. The results presented in this paper offer facilities, particularly locality and transitivity properties, to treat submodel issues in model transformation.

Submodels and Model Exchange

Collaborative modelling [45] is also an area where submodels appear to be useful. In collaborative modeling, the work is typically carried out by several actors, e.g. designers, automatic model processing applications, during the process and it is often the case that actors need to manipulate only some specific parts of a whole model simultaneously, possibly in remote locations. Communicating whole models and letting users and their tools operate on unrestricted areas of shared models is not desirable for performance and security reasons. Indeed, using model fragments to support such collaborative modeling activities appears more appropriate as a unit of distribution since it gives the ability to reduce the amount of communicated data and enables a way to control manipulation of models concurrently. The use of model fragment in this area has been investigated in [44]. In this work, the authors discuss a model bus that supports the interoperability between modeling tools and enables them to share their services and models using remote call mechanisms where transmitted parameters are model fragment instead of complete models. To define a model fragment in this approach, a user selects model elements or uses a helper operation that recursively computes a group of hierarchical elements with respect to the aggregation relationships of the related metamodel. Compared to our work, the notion of model fragment proposed in this approach is not formalized and is more restrictive than our notion of submodel since it only considers structural constraints based on the aggregation relationships. Our work could contribute advantageously to this area on two main points. First, the notion of model fragment could be substituted with our notion of submodel to improve the consistency of model fragments transmitted as parameter. Second, the operation of transmitting a consistent model fragment could be enhanced with the definition of our extraction operator.

8 Conclusion

In Model-Driven Engineering, the capacities of handling, comparing, extracting or delimiting proper submodels or parts (also called fragments or slices in other works) of a model are at the core of many activities, for example, model management through model repositories which was presented as a typical situation and application. Despite the obvious importance of the submodel notion and its corresponding relationship in MDE practices, few works were dedicated to their characterization and systematic investigation of their properties. It is the intent of this paper to contribute to this need.

For this, we started from a formal definition of a model as a set of model elements *plus* a set of dependency constraints that it asserts over these elements. This original formulation is simple and powerful enough to define models, submodels or any fragment of a model in order to precisely characterize their inclusion relationships. From this setting, we isolated the concepts of *closed*, *covariant* and *invariant* submodels which lead to interesting properties that we stated. In particular we showed that submodel transitivity can be guaranteed thanks to the notion of *submodel invariance*. This work is the core of the paper and lays algebraic basis for deeper understanding of the submodel notion.

This formalism offers keys to analyze and characterize operations which manipulate submodels. For example, we analyzed the operator which consists in extracting a model from another one when selecting some subset of its elements. Thanks to the preceding formalization, its properties were systematically examined. This allows to precisely position the produced models as qualified submodels compared to overall models under consideration.

The same can be applied to many other model operations. The last part of the paper is dedicated to a synthesis on related works and MDE practices which make

directly use or resort to the submodel notion. As far as engineering is concerned, stated properties are related to many expected qualities, such as locality, modularity and transitivity of model processing operation. As mentioned in this section, we ourselves do use the submodel notion in our works on view-oriented and template-based modeling where expected final models appear to be assemblies of submodels. Let us recall that the present stating comes from a generalization of these experiences. Backwardly, thanks to this groundings, we are systematically formalizing these practices and qualifying engaged submodels.

More generally, presented results can be seen as a foundation for further investigation on submodel operation studied elsewhere. In the same last section, we examined related works, compared to our contribution. It appears that many acceptances of the submodel notion (due, for example, to Model Composition or conversely Model Extraction, Model Compatibility) could advantageously profit of the results presented in this paper in order to better characterize practices and produced submodels. As a conclusion, we hope that this work will help in the quest for a better structuring of the MDE "model space" [27].

References

1. MDA. Home Page. http://www.omg.org/mda, 1997.
2. M. Acher, P. Collet, P. Lahire, and R.B. France. Slicing feature models. In *Proceedings of 26th International Conference on Automated Software Engineering (ASE'11)*, pages 424–427. IEEE/ACM, 2011.
3. M. Alanen and I. Porres. Difference and Union of Models. In *Proceedings of 6th International Conference on the Unified Modeling Language, Modeling Languages and Applications (UML'03)*, volume 2863 of *LNCS*, pages 2–17. Springer, 2003.
4. B. K. Appukuttan, T. Clark, A. Evans, G. Maskeri, P. Sammut, L. Tratt, and J. S. Willans. A pattern based approach to defining the dynamic infrastructure of UML 2.0. Technical report, March 2002.
5. J. H. Bae, K. Lee, and H. Seok Chae. Modularization of the UML Metamodel Using Model Slicing. In *Proceedings of 5th International Conference on Information Technology: New Generations (ITNG'08)*, pages 1253–1254. IEEE Computer Society, 2008.
6. E. Baniassad and S. Clarke. Theme: An Approach for Aspect-Oriented Analysis and Design. In *Proceedings of 26th International Conference on Software Engineering (ICSE '04)*, pages 158–167. IEEE Computer Society, 2004.
7. M. Barbero and J. Bézivin. Structured Libraries of Models. In *Proceedings of 1st International Workshop on Towers of Models (TOWERS'07)*, 2007.
8. J. Bézivin, F. Jouault, P. Rosenthal, and P. Valduriez. Modeling in the Large and Modeling in the Small. In *Model Driven Architecture*, volume 3599 of *LNCS*. Springer, 2005.
9. A. Blouin, B. Combemale, B. Baudry, and O. Beaudoux. Modeling Model Slicers. In *Proceedings of 14th International Conference on Model Driven Engineering Languages and Systems (MoDELS'11)*, volume 6981 of *LNCS*, pages 62–76. Springer, 2011.
10. O. Caron, B. Carré, A. Muller, and G. Vanwormhoudt. A Framework for Supporting Views in Component Oriented Information Systems. In *Proceedings of 9th International Conference on Object-Oriented Information Systems (OOIS'03)*, volume 2817 of *LNCS*, pages 164–178. Springer, 2003.
11. O. Caron, B. Carré, A. Muller, and G. Vanwormhoudt. An OCL Formulation of UML 2 Template Binding. In *Proceedings of 7th International Conference on The Unified Modeling Language, Model Languages and Applications (UML'04)*, volume 3273 of *LNCS*, pages 27–40. Springer, 2004.
12. O. Caron, B. Carré, A. Muller, and G. Vanwormhoudt. A Coding Framework for Functional Adaptation of Coarse-Grained Components in Extensible EJB Servers. In *Proceedings of Objects, Components, Models and Patterns, 47th International Conference (TOOLS EUROPE'09)*, volume 33 of *LNBIP*, pages 215–230. Springer, 2009.
13. T. Clark, A. Evans, and S. Kent. Aspect-oriented metamodelling. *Computer Journal*, 46(5):566–577, 2003.
14. S. Clarke. Extending standard UML with Model Composition Semantics. In *Science of Computer Programming*, volume 44, pages 71–100. Elsevier Science, 2002.
15. J. Dingel, Z. Diskin, and A. Zito. Understanding and improving UML package merge. *Software and System Modeling*, 7(4):443–467, 2008.
16. D. D'Souza and A. Wills. *Objects, Components and Frameworks With UML: The Catalysis Approach.* Addison-Wesley, 1999.
17. T. Elrad, O. Aldawud, and A. Bader. Aspect-Oriented Modeling: Bridging the Gap between Implementation and Design. In *Proceedings of 1st conference on Generative Programming and Component Engineering (GPCE '02)*, volume 2487, pages 189–201. Springer, 2002.
18. A. Etien, A. Muller, T. Legrand, and X. Blanc. Combining independent model transformations. In *Proceedings of 2010 ACM Symposium on Applied Computing (SAC'10)*, pages 2237–2243. ACM, 2010.
19. R. B. France, J. M. Bieman, and B. H. C. Cheng. Repository for Model Driven Development (ReMoDD). In *Proceeding of MoDELS'06 Workshops*, volume 4364 of *LNCS*, pages 311–317. Springer, 2006.
20. R. B. France, G. Georg, and I. Ray. Supporting Multi-Dimensional Separation of Design Concerns. In *Proceedings of AOSD Workshop on AOM: Aspect-Oriented Modeling with UML*, march 2003.
21. Object Management Group. Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT). http://www.omg.org/spec/QVT/.
22. C. Guy, B. Combemale, S. Derrien, J. Steel, and J.M. Jézéquel. On Model Subtyping. In *Proccedings of 8th European Conference on Modelling Foundations and Applications (ECMFA 2012)*, volume 7349 of *LNCS*, pages 400–415. Springer, 2012.
23. C. Jeanneret, M. Glinz, and B. Baudry. Estimating footprints of model operations. In *Proceedings of 33rd International Conference on Software Engineering (ICSE'11)*, pages 601–610. ACM, 2011.

24. F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A model transformation tool. In *Science of Computer Programming*, volume 72, pages 31–39. Elsevier Science, 2008.
25. H. H. Kagdi, J. I. Maletic, and A. Sutton. Context-Free Slicing of UML Class Models. In *Proceedings of 21st IEEE International Conference on Software Maintenance (ICSM'05)*, pages 635–638. IEEE Computer Society, 2005.
26. P. Kelsen, Q. Ma, and C. Glodt. Models within Models: Taming Model Complexity Using the Sub-model Lattice. In *Proceedings of 14th International Conference on Fundamental Approaches to Software Engineering, FASE'11*, volume 6603 of *LNCS*, pages 171–185. Springer, 2011.
27. S. Kent. Model Driven Engineering. In *Proceedings of the 3rd International Conference on Integrated Formal Methods (IFM'02)*, volume 2335 of *LNCS*, pages 286–298. Springer-Verlag, May 2002.
28. J. Klein, L. Hélouët, and J. M. Jézéquel. Semantic-based Weaving of Scenarios. In *Proceedings of 5th International Conference on Aspect-Oriented Software Development (AOSD'06)*, pages 27–38. ACM, 2006.
29. B. Korel, I. Singh, L. H. Tahat, and B. Vaysburg. Slicing of State-Based Models. In *Proceedings of 19th International Conference on Software Maintenance (ICSM'03)*, pages 34–43. IEEE Computer Society, 2003.
30. T. Kuhne. An Observer-Based Notion of Model Inheritance. In *Proceedings of 13th International Conference on Model Driven Engineering Languages and Systems (MoDELS'10)*, volume 6394 of *LNCS*, pages 31–45. Springer, 2010.
31. Ph. Lahire, B. Morin, G. Vanwormhoudt, A. Gaignard, O. Barais, and J-M Jézéquel. Introducing Variability into Aspect-Oriented Modeling Approaches. In *Proceedings of 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS'07)*, volume 4735 of *LNCS*, pages 498–513. Springer, 2007.
32. K. Lano and S. Kolahdouz-Rahimi. Slicing Techniques for UML Models. *Journal of Object Technology*, 10:11:1–49, 2011.
33. T. Levendovszky, L. Lengyel, and T. Mészáros. Supporting domain-specific model patterns with metamodeling. *Software and Systems Modeling*, 8:501–520, 2009.
34. D. Lucrédio, R. Pontin de Mattos Fortes, and J. Whittle. Moogle: a metamodel-based model search engine. *Software and System Modeling*, 11(2):183–208, 2012.
35. S. Melnik, E. Rahm, and Ph. A. Bernstein. Rondo: a programming platform for generic model management. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 193–204. ACM, 2003.
36. A. Muller, O. Caron, B. Carré, and G. Vanwormhoudt. On Some Properties of Parameterized Model Application. In *Proceedings of 1st European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA'05)*, volume 3748 of *LNCS*, pages 130–144. Springer, November 2005.
37. S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave. Matching and Merging of Statecharts Specifications. In *Proceedings of 29th international conference on Software Engineering (ICSE'07)*, pages 54–64. IEEE Computer Society, 2007.
38. Y. R. Reddy, S. Ghosh, R. B. France, G. Straw, J. M. Bieman, N. McEachen, E. Song, and G. Georg. Directives for Composing Aspect-Oriented Design Class Models. *Transactions on Aspect-Oriented Software Development I*, 3380:75–105, 2006.
39. T. Reiter, E. Kapsammer, W. Retschitzegger, and W. Schwinger. Model Integration Through Mega Operations. In *Proceedings of the International Workshop on Model-driven Web Engineering (MDWE'05)*, 2005.
40. S. Sen, N. Moha, B. Baudry, and J-M. Jézéquel. Metamodel Pruning. In *Proceedings of 12th International Conference on Model Driven Engineering Languages and Systems (MoDELS'09)*, volume 5795 of *LNCS*, pages 32–46. Springer, 2009.
41. S. Sen, N. Moha, V. Mahé, O. Barais, B. Baudry, and J-M. Jézéquel. Reusable model transformations. *Software and Systems Modeling*, 2010.
42. M. Siikarla, J. Peltonen, and J. Koskinen. Towards unambiguous model fragments. *Nordic Journal of Computing*, 13:180–195, 2006.
43. A. Solberg, R. France, and R. Reddy. Navigating the MetaMuddle. In *Proceedings of 4th Workshop in Software Model Engineering*, 2005.
44. P. Sriplakich, X. Blanc, and M-P. Gervais. Applying Model Fragment Copy-Restore to Build an Open and Distributed MDA Environment. In *Proceedings of 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS'06)*, volume 4199 of *LNCS*, pages 631–645. Springer, 2006.
45. P. Sriplakich, X. Blanc, and M-P. Gervais. Supporting Collaborative Development in an Open MDA Environment. In *Proceedings of 22nd IEEE International Conference on Software Maintenance (ICSM'06)*, pages 244–253. IEEE Computer Society, 2006.
46. P. Sriplakich, X. Blanc, and M.P. Gervais. Collaborative software engineering on large-scale models: requirements and experience in modelbus. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 674–681. ACM, 2008.
47. J. Steel and J-M. Jézéquel. On model typing. *Software and System Modeling*, 6(4):401–413, 2007.
48. G. Taentzer, C. Ermel, P. Langer, and M. Wimmer. A fundamental approach to model versioning based on graph modifications: from theory to implementation. *Software and Systems Modeling*, pages 1–34, 2012.
49. Auxiliary Constructs Templates, UML 2.0 Superstructure Specification. http://www.omg.org/spec/UML, 2003.
50. J. Whittle, P. K. Jayaraman, A. M. Elkhodary, A. Moreira, and J. Araújo. MATA: A Unified Approach for Composing UML Aspect Models Based on Graph Transformation. *Transactions on Aspect-Oriented Software Development VI*, 6:191–237, 2009.