A Generic Model Decomposition Technique

Pierre Kelsen and Qin Ma and Christian Glodt
Laboratory for Advanced Software Systems
University of Luxembourg
6, rue R. Coudenhove-Kalergi, Luxembourg

TR-LASSY-10-06

Abstract

Model-driven software development aims at easing the process of software development by using models as primary artifacts. Although less complex than the real systems they are based on, models tend to be complex nevertheless, thus making the tasks such as model management, model maintenance, and model comprehension non-trivial in many cases. In this paper we propose a generic model decomposition technique to tackle the complexity. This technique decomposes models into sub-models that conform to the same metamodel as the original model. The main contributions of this paper are: a mathematical description of the structure of these sub-models as a lattice, a linear-time algorithm for constructing this decomposition and finally two applications of our decomposition technique to model comprehension.

1 Introduction

In model-driven software development models are the primary artifacts. Typically several models are used to describe the different concerns of a system. One of the main motivations for using models is the problem of dealing with the

complexity of real systems: because models represent abstractions of a system, they are typically less complex than the systems they represent.

Nevertheless models for real systems can be complex themselves and thus may require aids for facilitating human comprehension. The problem of understanding complex models is at the heart of this paper. We propose a method for decomposing models that is based on subdividing models into smaller sub-models with the property that these sub-models conform to the same metamodel as the original model. This property allows to view the sub-models using the same tools as the original model and to understand the meaning of the sub-models using the same semantic mapping (if one has been defined).

An example of a concrete application scenario is the following: when trying to understand a large model, one starts with a subset of concepts that one is interested in (such as the concept of Class in the UML metamodel). Our method allows to construct a small sub-model of the initial model that contains all entities of interest and that conforms to the original metamodel (in the case of UML this would be MOF). The latter condition ensures that the sub-model can be viewed in the same way as the original model and that it has a well-defined semantics. The smaller size (compared to the original model) should facilitate comprehension.

The main contributions of this paper are the following: (a) we present the mathematical structure of these sub-models as a lattice, with the original model at the top and the empty sub-model at the bottom; (b) we present a linear time algorithm for building a decomposition hierarchy for a model from which the sub-model lattice can be constructed in a straightforward manner; (c) we present a method for the user to comprehend models in the context of model pruning.

One salient feature of our technique is its generic nature: it applies to any model (even outside the realm of model-driven software development). This is also one of the main features differentiating it from existing work in model decomposition. We review here related work from model slicing, metamodel pruning and model abstraction.

The idea behind model slicing is to generalize the work on program slicing to the domains of models by computing parts of models that contain modeling elements of interest. An example of this line of work is [4] where model slicing of UML class diagrams is investigated. Another example is [2] which considers the problem of slicing the UML metamodel into metamodels corresponding to the different diagram types in UML. The main differences between our work and research on model slicing is, first, the restriction of model slicing to a particular modeling language, e.g. UML class diagrams, and, second, the focus on a single model rather than the mathematical structure of sub-models of interest.

In a similar line of work some authors have investigated the possibility of pruning metamodels in order to make them more manageable. The idea is to remove elements from a metamodel to obtain a minimal set of modeling elements containing a given subset of elements of interest. Such an approach is described in [8]. This work differs from our work in several respects: first, just like model slicing it focuses on a single model rather than considering the collection of relevant sub-models in its totality; second, it is less generic in the sense that

it restricts its attention to Ecore metamodels (and the pruning algorithm they present is very dependent on the structure of Ecore), and lastly their goal is not just to get a conformant sub-model but rather to find a sub-metamodel that is a supertype of the original model. This added constraint is due to main use of the sub-metamodel in model transformation testing.

The general idea of simplifying models (which can be seen as a generalization of model slicing and pruning) has also been investigated in the area of model abstraction (see [3] for an overview). In the area of simulation model abstraction is a method for reducing the complexity of a simulation model while maintaining the validity of the simulation results with respect to the question that the simulation is being used to address. Work in this area differs from ours in two ways: first, model abstraction techniques generally transform models and do not necessarily result in sub-models; second, conformance of the resulting model with a metamodel is not the main concern but rather validity of simulation results.

The remainder of this paper is structured as follows: in the next section we present formal definitions for models, metamodels and model conformance. In section 3 we describe an algorithm for decomposing a model into sub-models conformant to the same metamodel as the original model. We also present the mathematical structure of these models, namely the lattice of sub-models. In section 5 we outline the application to model comprehension and we present concluding remarks in the final section.

2 Models and MetaModels

In this section we present formal definitions of models, metamodels, and model conformance. The following notational conventions will be used:

1. For any tuple p, we use $\mathsf{fst}(p)$ to denote its first element, and $\mathsf{snd}(p)$ to denote its second element.
2. For any set s, we use $\sharp s$ to denote its cardinality.
3. We use $\leq$ to denote the inheritance based subtyping relation.

2.1 Metamodels

A metamodel defines (the abstract syntax of) a language for expressing models. It consists of a finite set of metaclasses and a finite set of relations between metaclasses - either associations or inheritance relations. Moreover, a set of constraints may be specified in the contexts of metaclasses as additional well-formedness rules.

Definition 1 (Metamodel)**.** A metamodel $\mathbb{M} = (\mathbb{N}, \mathbb{A}, \mathbb{H}, \mathbb{C})$ is a tuple:

- $\mathbb{N}$ is the set of metaclasses, and $\mathbb{n} \in \mathbb{N}$ ranges over it.

- $\mathbb{A} \subseteq (\mathbb{N} \times \mu) \times (\mathbb{N} \times \mu \times \mathbb{R})$ represents the (directed) associations between metaclasses and $\mathbb{a} \in \mathbb{A}$ ranges over it. The two $\mathbb{N}$'s give the types of the two association ends. The two μ's, where $\mu \in Int \times \{Int \cup \{\infty\}\}$, give the corresponding multiplicities. We refer to the first end of the association as the source, and the second as the target. Associations are navigable from source to target, and the navigation is represented by referring to the given role name that is attached to the target end selected from the vocabulary $\mathbb{R}$ of role names.

- $\mathbb{H} \subseteq \mathbb{N} \times \mathbb{N}$ denotes the inheritance relation among metaclasses and $\mathbb{h} \in \mathbb{H}$ ranges over it. For a given $\mathbb{h} \in \mathbb{H}$, $\mathsf{fst}(\mathbb{h})$ inherits from (i.e., is a subtype of) $\mathsf{snd}(\mathbb{h})$.

- $\mathbb{C} \subseteq \mathbb{N} \times \mathbb{E}$ gives the set of constraints applied to the metamodel and $\mathbb{c} \in \mathbb{C}$ ranges over it. $\mathbb{E}$ is the set of the expressions and $\mathbb{e} \in \mathbb{E}$ ranges over it. A constraint $\mathbb{c} = (\mathbb{n}, \mathbb{e})$ makes the context metaclass $\mathbb{n}$ more precise by restricting it with an assertion, i.e., the boolean typed expression $\mathbb{e}$. For example, a constraint can further restrict the multiplicities or types of association ends that are related to the context metaclass.

Let $\mathbb{r} \in \mathbb{R}$ range over the set of role names mentioned above. We require that role names in a metamodel are distinct. As a consequence, it is always possible to retrieve the association that corresponds to a given role name, written $\mathsf{asso}(\mathbb{r})$.

2.2 Models

A model is expressed in a metamodel. It is built by instantiating the constructs, i.e., metaclasses and associations, of the metamodel.

Definition 2 (Model)**.** A model is defined by a tuple $\mathsf{M} = (\mathbb{M}, \mathsf{N}, \mathsf{A}, \tau)$ where:

- $\mathbb{M}$ is the metamodel in which the model is expressed.

- N is the set of metaclass instantiations of the metamodel $\mathbb{M}$, and $\mathsf{n} \in \mathsf{N}$ ranges over it. They are often simply referred to as instances when there is no possible confusion.

- $\mathsf{A} \subseteq \mathsf{N} \times (\mathsf{N} \times \mathbb{R})$ is the set of association instantiations of the metamodel $\mathbb{M}$, and $\mathsf{a} \in \mathsf{A}$ ranges over it. They are often referred to as links.

- τ is the typing function: $(\mathsf{N} \to \mathbb{N}) \cup (\mathsf{A} \to \mathbb{A})$. It records the type information of the instances and links in the model, i.e., from which metaclasses or associations of the metamodel $\mathbb{M}$ they are instantiated.

2.3 Model conformance

Not all models following the definitions above are valid, or "conform to" the metamodel: typing, multiplicity, and constraints need all to be respected.

Definition 3 (Model conformance)**.** We say a model $\mathsf{M} = (\mathbb{M}, \mathsf{N}, \mathsf{A}, \tau)$ conforms to its metamodel $\mathbb{M}$ or is valid when the following conditions are met:

1. type compatible:

   $$\forall \mathsf{a} \in \mathsf{A}, \tau(\mathsf{fst}(\mathsf{a})) \leq \mathsf{fst}(\mathsf{fst}(\tau(\mathsf{a}))) \text{ and } \tau(\mathsf{fst}(\mathsf{snd}(\mathsf{a}))) \leq \mathsf{fst}(\mathsf{snd}(\tau(\mathsf{a})))$$

   Namely, the types of the link ends must be compatible with (being subtypes of) the types as specified in the corresponding association ends.

2. multiplicity compatible: $\forall \mathsf{n} \in \mathsf{N}, \mathbb{a} \in \mathbb{A}$,

   if $\tau(\mathsf{n}) \leq \mathsf{fst}(\mathsf{fst}(\mathbb{a}))$,

   then $\sharp\{\mathsf{a} \mid \mathsf{a} \in \mathsf{A} \text{ and } \tau(\mathsf{a}) = \mathbb{a} \text{ and } \mathsf{fst}(\mathsf{a}) = \mathsf{n}\} \in \mathsf{snd}(\mathsf{snd}(\mathbb{a}))$;

   if $\tau(\mathsf{n}) \leq \mathsf{fst}(\mathsf{snd}(\mathbb{a}))$,

   then $\sharp\{\mathsf{a} \mid \mathsf{a} \in \mathsf{A} \text{ and } \tau(\mathsf{a}) = \mathbb{a} \text{ and } \mathsf{fst}(\mathsf{snd}(\mathsf{a})) = \mathsf{n}\} \in \mathsf{snd}(\mathsf{fst}(\mathbb{a}))$.

   Namely, the number of link ends should conform to the specified multiplicity in the corresponding association end.

3. constraints satisfied: $\forall \mathbb{c} \in \mathbb{C}$, $\forall \mathsf{n} \in \mathsf{N}$ where n is an instance of the context metaclass, i.e., $\tau(\mathsf{n}) \leq \mathsf{fst}(\mathbb{c})$, the boolean expression $\mathsf{snd}(\mathbb{c})$ should evaluate to true in model M for the contextual instance n.

3 Model Decomposition

3.1 Criteria

Model decomposition starts from a model that conforms to a metamodel, and decomposes it into smaller parts. Our model decomposition technique is designed using the following as main criterion: the derived parts should be valid models conforming to the original metamodel. Achieving this goal has two main advantages:

1. the derived parts, being themselves valid models, can be comprehended on their own according to the familiar abstract syntax and semantics (if defined) of the modeling language;

2. the derived parts can be wrapped up into modules and reused in the construction of other system models, following our modular model composition paradigm [5].

The decomposed smaller parts of a model are called its *sub-models*, formally defined below.

Definition 4 (Sub-model)**.** We say a model $\mathsf{M}' = (\mathbb{M}, \mathsf{N}', \mathsf{A}', \tau')$ is a sub-model of another model $\mathsf{M} = (\mathbb{M}, \mathsf{N}, \mathsf{A}, \tau)$ if and only if:

1. $\mathsf{N}' \subseteq \mathsf{N}$;

2. $\mathsf{A}' \subseteq \mathsf{A}$;

3. τ' is a restriction of τ to N' and A'.

In order to make the sub-model M' also conform to $\mathbb{M}$, we will propose three conditions - one for the metamodel (Condition 3 below, regarding the nature of the constraints) and two conditions for the sub-model (Conditions 1 and 2). Altogether these three conditions will be sufficient to ensure conformance of the sub-model.

The starting point of our investigation is the definition of conformance (Definition 3). Three conditions must be met in order for sub-model M' to conform to metamodel $\mathbb{M}$.

The first condition for conformance, type compatibility, follows directly from the fact that M' is a sub-model of M and M conforms to $\mathbb{M}$. The second condition for conformance, which we call the *multiplicity condition*, concerns the multiplicities on the association ends in the metamodel $\mathbb{M}$. First the number of links ending at an instance of M' must agree with the source cardinality of the corresponding association in the metamodel and second the number of links leaving an instance of M' must agree with the target cardinality of the corresponding association.

To ensure the multiplicity condition for links ending at an instance of the sub-model, we will introduce the notion of *fragmentable links*, whose type (i.e., the corresponding association) has an un-constrained (i.e., being 0) lower bound for the source cardinality.

Definition 5 (Fragmentable link)**.** Given a model $\mathsf{M} = (\mathbb{M}, \mathsf{N}, \mathsf{A}, \tau)$, a link $\mathsf{a} \in \mathsf{A}$ is fragmentable if $\mathsf{snd}(\mathsf{fst}(\tau(\mathsf{a}))) = (0, \_)$, where $\_$ represents any integer whose value is irrelevant for this definition.

Fragmentable incoming links of M to instances in M' are safe to exclude but this is not the case for non-fragmentable links, which should all be included. We thus obtain the first condition on sub-model M':

Condition 1. $\forall \mathsf{a} \in \mathsf{A}$ *where* a *is non-fragmentable,* $\mathsf{fst}(\mathsf{snd}(\mathsf{a})) \in \mathsf{N}'$ *implies* $\mathsf{fst}(\mathsf{a}) \in \mathsf{N}'$ *and* $\mathsf{a} \in \mathsf{A}'$.

Let us now consider the multiplicity condition for links leaving an instance of M'. To ensure this condition we shall require that M' includes all the links of M that leave an instance of M'. In other words, there are in fact no links leaving M'. This is formally expressed in the following condition on the sub-model:

Condition 2. $\forall \mathsf{a} \in \mathsf{A}$, $\mathsf{fst}(\mathsf{a}) \in \mathsf{N}'$ *implies* $\mathsf{fst}(\mathsf{snd}(\mathsf{a})) \in \mathsf{N}'$ *and* $\mathsf{a} \in \mathsf{A}'$.

Conditions 1 and 2 together imply the multiplicity condition in the conformance definition.

The third condition in the conformance definition, which we call the *constraint condition*, requires that all metamodel constraints are satisfied in sub-model M'. These constraints are known satisfied in model M because M conforms

to the metamodel. Therefore, we need to maintain the evaluation of the boolean expressions in these constraints while sub-modeling from M to M'.

Let us first introduce some knowledge about constraint evaluation. We write $\mathsf{eval}(\mathbb{c}, \mathsf{M}, \mathsf{n})$ for the evaluation of (the boolean expression of) a constraint $\mathbb{c}$ in a model M for a contextual instance n. An evaluation is well-formed if $\mathbb{c}$ is defined for the metamodel of M, n is an instance of M, and the type of n is a subtype of the context metaclass of $\mathbb{c}$. The result of a well-formed constraint evaluation is determined by its *scope*, defined below.

Definition 6 (Scope of constraint evaluation)**.** The scope of a well-formed evaluation $\mathsf{eval}(\mathbb{c}, \mathsf{M}, \mathsf{n})$ is a pair $(\mathsf{M}_S, \mathsf{n})$, where M_S is a sub-model of M consisting only of all the model elements, instances and links, referenced by the boolean expression of $\mathbb{c}$.

For example, the mainstream language for specifying MOF metamodel constraint expressions: EssentialOCL [7], basically provides two ways: model elements are referenced by either starting from the contextual instance and following association navigations, hence are all reachable from the contextual instance; or using the $\mathsf{AllInstances}$ operation to reference all the instances of a given metaclass.

In addition, we have the following property of constraint evaluation that is relevant for this paper:

Property 1. $\mathsf{eval}(\mathbb{c}, \mathsf{M}, \mathsf{n}) = \mathsf{eval}(\mathbb{c}, \mathsf{M}', \mathsf{n})$ *if* M' *is a sub-model of* M *and contains the scope of* $\mathsf{eval}(\mathbb{c}, \mathsf{M}, \mathsf{n})$.

In other words, parts of M outside the scope are safe to exclude without any influence on the constraint evaluation. Subsequently, to maintain scopes becomes the key to maintain constraint satisfaction while sub-modeling M.

Because of Condition 2, the sub-model M' includes all the reachable model elements in M from the contextual instance. We impose a restriction on the nature of the constraints by introducing the notion of *forward constraints*.

Definition 7 (Forward constraint)**.** A constraint is forward if for any well-formed evaluation $\mathsf{eval}(\mathbb{c}, \mathsf{M}, \mathsf{n})$, the corresponding scope $(\mathsf{M}_S, \mathsf{n})$ is reachable from n, namely:

1. all instances of M_S are reachable from n: $\forall \mathsf{n}_S \in \mathsf{N}_S$, $\exists \mathsf{a}_i \in \mathsf{A}$, $1 \leq i \leq k$, s.t. $\mathsf{fst}(\mathsf{snd}(a_i)) = \mathsf{fst}((a_{i+1}))$, $1 \leq i \leq k-1$, $\mathsf{fst}(a_1) = \mathsf{n}$ and $\mathsf{fst}(\mathsf{snd}(a_k)) = \mathsf{n}_S$;
2. all links of M_S are reachable from n: $\forall \mathsf{a}_S \in \mathsf{A}_S$, $\mathsf{fst}(\mathsf{a}_S)$ (i.e. the source instance of the link) is reachable from n.

As a consequence, to ensure the constraint condition, it is sufficient to only allow forward constraints. This is expressed in the following condition over the metamodel $\mathbb{M}$:

Condition 3. *All constraints in metamodel* $\mathbb{M}$ *are forward constraints.*

Note that we formalize a core part of the EssentialOCL [7] in section 4, which in principal excludes AllInstances, called CoreOCL. We precisely define the notion of constraint evaluation and scope for CoreOCL, demonstrate Properpty 1, and prove that all CoreOCL constraints are forward.

It is not difficult to see that both Conditions 2 and 3 imply the constraint condition in the conformance definition. Indeed Condition 2 implies that all instances and links reachable from an instance n in model M are also reachable in M' if M' does indeed contain n. Condition 3 then implies that a constraint that is satisfied on contextual instance n in M is also satisfied in the sub-model M' since it references the same instances and links in both models.

We thus obtain the following result:

Theorem 1. *Given a metamodel* $\mathbb{M}$*, a model* M*, and a sub-model* M' *of* M*, suppose that:*

1. *model* M *conforms to* $\mathbb{M}$*;*
2. *model* M' *satisfies Condition 1 and 2;*
3. *metamodel* $\mathbb{M}$ *satisfies Condition 3;*

then model M' *also conforms to the metamodel* $\mathbb{M}$*.*

Proof. The result follows from the discussion above. □

3.2 Algorithm

From hereon we shall assume that the metamodel under consideration satisfies Condition 3. In this subsection we describe an algorithm that finds, for a given model M, a decomposition of M such that any sub-model of M that satisfies both Condition 1 and 2 can be derived from the decomposition by uniting some components of the decomposition.

We reach the goal in two steps: (1) ensure Condition 1 and 2 with respect to only non-fragmentable links; (2) ensure again Condition 2 with respect to fragmentable links. (Condition 1 does not need to be re-assured because it only involves non-fragmentable links.) Details of each step are discussed below.

Treating instances as vertices and links as edges, models are just graphs. For illustration purpose, consider an example model as presented in Figure 1 where all fragmentable links are indicated by two short parallel lines crossing the links.

Let G be the graph derived by removing the fragmentable links from M. Because all the links in G are non-fragmentable, for a sub-graph of G to satisfy both Condition 1 and 2, an instance is included in the sub-graph if and only if all its ancestor and descendant instances are also included, and so are the links among these instances in G. These instances, from the point of view of graph theory, constitute a weakly connected component (wcc) of graph G (i.e., a connected component if we ignore edge directions). The first step of the model decomposition computes all such wcc's of G, which disjointly cover all

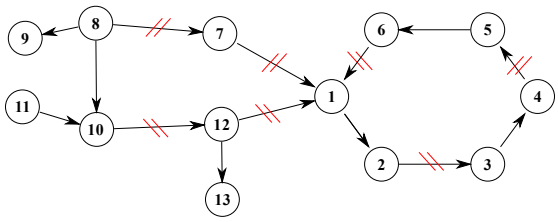

Figure 1: An example model

the instances in model M, then puts back the fragmentable links. We collapse all the nodes that belong to one wcc into one node, (referred to as *a wcc-node* in contrast to the original nodes), and refer to the result as graph W. After the first step, the corresponding graph W of the example model contains six wcc-nodes inter-connected by fragmentable links, as shown in Figure 2.

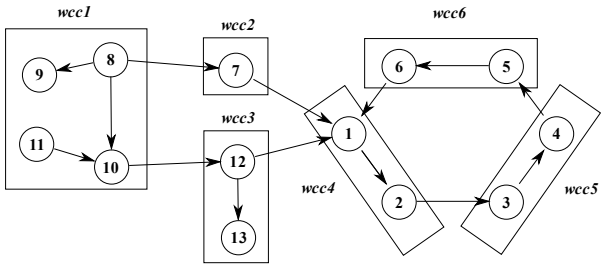

Figure 2: The corresponding W graph of the example model after step 1

The instances and links that are collapsed into one wcc-node in W constitute a sub-model of M satisfying both Condition 1 and 2, but only with respect to non-fragmentable links, because wcc's are computed in the context of G where fragmentable links are removed. The second step of the model decomposition starts from graph W and tries to satisfy Condition 2 with respect to fragmentable links, i.e., following outgoing fragmentable links. More specifically, we compute all the strongly connected components (scc's) in W (see [10] for a definition of strongly connected components) and collapse all the nodes that belong to one scc into one node, (referred to as *an scc-node*), and refer to the result as graph D. After the second step, the corresponding graph D of the example model looks like in Figure 3. The three wcc-nodes $wcc4$, $wcc5$ and $wcc6$ of graph W are collapsed into one scc-node $scc4$ because they lie on a (directed) cycle.

Note that we only collapse nodes of a strongly connected component in the second step instead of any reachable nodes following outgoing fragmentable links in W, because we do not want to loose any potential sub-model of M satisfying both Condition 1 and 2 on the way. More precisely, a set of nodes is collapsed

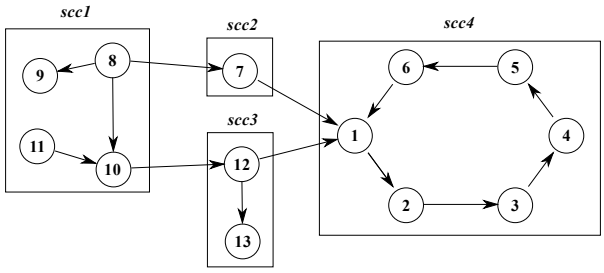

Figure 3: The corresponding D graph of the example model after step 2

only if for every sub-model M' of M satisfying both Condition 1 and 2, it is either completely contained in M' or disjoint with M', i.e., no such M' can tell the nodes in the set apart.

The computational complexity of the above algorithm is dominated by the complexity of computing weakly and strongly connected components in the model graph. Computing weakly connected components amounts to computing connected components if we ignore the direction of the edges. We can compute connected components and strongly connected components in linear time using depth-first search [10]. Thus the overall complexity is linear in the size of the model graph.

3.3 Correctness

Graph D obtained at the end of the algorithm is a DAG (Directed Acyclic Graph) with all the edges being fragmentable links. Graph D represents a decomposition of the original model M where all the instances and links that are collapsed into an scc-node in D constitute a component in the decomposition. We call graph D *the decomposition hierarchy* of model M.

To relate the decomposition hierarchy to the sub-models, we introduce the concept of an *antichain-node*. An antichain-node is derived by collapsing a (possibly empty) antichain of scc-nodes (i.e., a set of scc-nodes that are neither descendants nor ancestors of one another, the concept of antichain being borrowed from order theory) plus their descendants (briefly an antichain plus descendants) in the decomposition hierarchy. To demonstrate the correctness of the algorithm, we prove the following theorem:

Theorem 2. *Given a model* $\mathsf{M} = (\mathbb{M}, \mathsf{N}, \mathsf{A}, \tau)$ *and a sub-model* $\mathsf{M}' = (\mathbb{M}, \mathsf{N}', \mathsf{A}', \tau)$ *of* M*,* M' *satisfies both Condition 1 and 2 if and only if there exists a corresponding antichain-node of the decomposition hierarchy of* M *where* M' *consists of the instances and links collapsed in this antichain-node.*

Proof. We first demonstrate that if M' consists of the set of instances and links that are collapsed in an antichain-node of the decomposition hierarchy of M, then M' satisfies both Condition 1 and 2.

- Check M' against Condition 1: given a non-fragmentable link $\mathsf{a} \in \mathsf{A}$, if $\mathsf{fst}(\mathsf{snd}(\mathsf{a})) \in \mathsf{N}'$, we have $\mathsf{fst}(\mathsf{a}) \in \mathsf{N}'$ because of the wcc computation in the first step of the model decomposition algorithm.

- Check M' against Condition 2: given a non-fragmentable link $\mathsf{a} \in \mathsf{A}$, if $\mathsf{fst}(\mathsf{a}) \in \mathsf{N}'$, we have $\mathsf{fst}(\mathsf{snd}(\mathsf{a})) \in \mathsf{N}'$ because of the wcc computation in the first step of the model decomposition algorithm. Given a fragmentable link $\mathsf{a} \in \mathsf{A}$, if $\mathsf{fst}(\mathsf{a}) \in \mathsf{N}'$, we have $\mathsf{fst}(\mathsf{snd}(\mathsf{a})) \in \mathsf{N}'$ because of the scc computation in the second step of the model decomposition algorithm and because we take all the descendants into account.

We now demonstrate the other direction of the theorem, namely, if M' satisfies both Condition 1 and 2, then there exists an antichain-node of the decomposition hierarchy of M, such that M' consists of the set of instances and links that are collapsed in this antichain-node.

We refer to the set of scc-nodes in the decomposition hierarchy where each includes at least one instance of M' by S.

1. All the instances that are collapsed in an scc-node in S belong to M'. Given an scc-node $s \in S$, there must exist an instance n collapsed in s and $\mathsf{n} \in \mathsf{N}'$ in order for s to be included in S. Let n' be another instance collapsed in s. Following the algorithm in subsection 3.2 to compute the decomposition hierarchy, n and n' are aggregated into one scc-node either in the first or the second step.

   (a) If they are aggregated in the first step, that means the two instances are weakly connected by non-fragmentable links, and because M' satisfies Condition 1 and 2, n' should also be in M'.

   (b) If they are aggregated in the second step but not in the first step, that means n and n' are aggregated in two separate wcc-nodes in the first step, called w and w', which are strongly connected by a path of fragmentable links. Referring to the other wcc-nodes on the path by $w_1, \ldots, w_k$, there exists a set of instances $\mathsf{n}_0 \in w$, $\mathsf{n}'_0 \in w'$, and $\mathsf{n}_i, \mathsf{n}'_i \in w_i$ for $1 \leq i \leq k$, such that there are fragmentable links from n_0 to n_1, from n'_i to n_{i+1} ($\forall i.1 \leq i < k$) and from n'_k to n'_0. Since M' satisfies Condition 2 if the source vertex of these fragmentable links belongs to M', so does the target vertex. Because the following pairs of instances: n and n_0, n_i and n'_i ($\forall i.1 \leq i \leq k$), and n'_0 and n', are respectively collapsed in a wcc-node, if one vertex in a pair belongs to M' then the other vertex in the pair must belong to M' as well following Condition 1 and 2. From the above discussion and by applying mathematical induction, n belonging to M' implies that n' belongs to M' as well.

2. S constitutes an antichain plus descendant. We partition S into two subsets: S_1 contains all the scc-nodes in S that do not have another scc-node also in S as ancestor; S_2 contains the rest, i.e., $S_2 = S \setminus S_1$. Clearly S_1

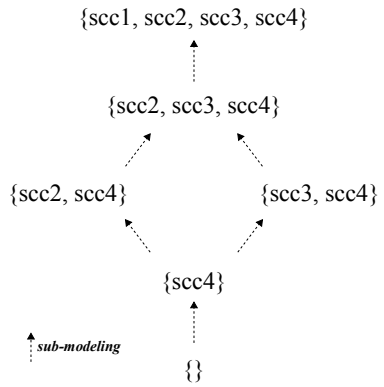

Figure 4: The sub-model lattice of the example model in Figure 1 whose decomposition hierarchy is given in Figure 2

constitutes an antichain, and any scc-node in S_2 is a descendant of an scc-node in S_1 because otherwise the former scc-node should belong to S_1 instead of S_2. Moreover, S_2 contains all the descendants of scc-nodes in S_1. Given a child s_2 of an scc-node $s_1 \in S_1$, the fragmentable link from s_1 to s_2 connects an instance n_1 collapsed in s_1 to an instance n_2 collapsed in s_2. Because $s_1 \in S_1$, following the demonstrated item 1 above, we have $\mathsf{n}_1 \in \mathsf{N}'$. Because of the out-going fragmentable link from n_1 to n_2 and since M' satisfies Condition 2, we also have $\mathsf{n}_2 \in \mathsf{N}'$. Therefore we have $s_2 \in S$. Furthermore, $s_2 \notin S_1$ because it has $s_1 \in S$ as its ancestor. Hence we have $s_2 \in S_2$. Inductively we conclude that any descendant of s_1 belongs to S_2 and hence S constitutes an antichain plus descendant.

3. Collapse all the scc-nodes in S into an antichain-node called A. We demonstrate that M' consists of the instances and links collapsed in A.

   (a) Any instance of M' is collapsed in A because of the selection criteria of S, and any instance collapsed in A is an instance of M' following the demonstrated item 1 above. In other words, M' and A contain the same set of instances from M.

   (b) Because both M' and A span all the links in M that connect instances in them, M' and A also have the same set of links from M.

□

3.4 The lattice of sub-models

Recall that a lattice is a partially-ordered set in which every pair of elements has a least upper bound and a greatest lower bound. Thanks to Theorem 2, we can now refer to a sub-model M' of model M that satisfies both Condition 1

and 2 by the corresponding antichain-node A in the decomposition hierarchy of M. Given a model M, all the sub-models that satisfy both Condition 1 and 2 constitute a lattice ordered by the relation "is a sub-model of", referred to as *the sub-model lattice* of M. Let A_1 and A_2 denote two such sub-models. The least upper bound ($A_1 \vee A_2$) and the greatest lower bound ($A_1 \wedge A_2$) of A_1 and A_2 are computed in the following way:

- $A_1 \vee A_2$ is the antichain-node obtained by collapsing the scc-nodes of A_1 and A_2;
- $A_1 \wedge A_2$ is the antichain-node obtained by collapsing the common scc-nodes of A_1 and A_2.

The top of the sub-model lattice is M itself, and the bottom is the empty sub-model.

For the example model discussed in subsection 3.2 whose decomposition hierarchy is given in Figure 3, six possible antichain-nodes can be derived from the decomposition hierarchy, denoted by the set of scc-nodes that are collapsed. They are ordered in a lattice as shown in Figure 4.

3.5 Implementation

We have implemented the model decomposition technique [1]. The implementation takes a model of any metamodel that follows Definition 1 as input, and computes the decomposition hierarchy of it from which the sub-model lattice can be constructed by enumerating all the antichain-nodes of the decomposition hierarchy. Note that in the worst case where the decomposition hierarchy contains no edges, the size of the sub-model lattice equals the size of the power-set of the decomposition hierarchy, which is exponential.

4 CoreOCL

In this section we formalize a subset of the OCL language [7] for the specification of constraint expressions attached to metamodels. We call the subset the CoreOCL. In a similar way as EssentialOCL, which is the subset of complete OCL required to work with EMOF, CoreOCL is the subset of EssentialOCL that is required for invariant specification in metamodels as defined in Definition 1. Constraint expressions written in CoreOCL in principal exclude the use of allInstances. In addtion, only the most fundamental and generic collection iteration operation iterate is included in order to keep the language minimal, as all the other collection iteration operations can be derived from it.

The rest of the section is organized as follows: the abstract syntax of CoreOCL is defined in subsection 4.1. The semantics of CoreOCL is defined in subsection 4.2 by precisely defining the evaluation of CoreOCL expressions and the corresponding scopes, upon which the definition of forward constraints given in Definition 7 relies. We then demonstrate Properpty 1. Finally, we prove that all CoreOCL constraints are forward in subsection 4.3.

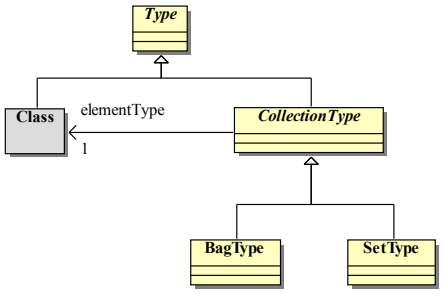

Figure 5: CoreOCL types: abstract syntax

4.1 Abstract syntax

Figure 5 and Figure 6 summarize the abstract syntax of CoreOCL. The detailed definition of CoreOCL expressions is also given below in EBNF format.

$\mathtt{e}$	::=	$\mathtt{i}$	Integer scalar
		$\mathtt{b}$	Boolean scalar
		$x \mid \mathsf{self}$	Variable
		$\mathtt{e}.\mathtt{r}$	Association end navigation
		$\kappa(\overline{\mathtt{e}})$	Built-in operation (kappa) call
		$\mathsf{let}\ x = \mathtt{e}_1\ \mathsf{in}\ \mathtt{e}_2$	Let binding
		$\mathsf{if}\ \mathtt{e}_1\ \mathsf{then}\ \mathtt{e}_2\ \mathsf{else}\ \mathtt{e}_3$	Conditional expression
		$\mathtt{e}\ \mathsf{asInstanceOf}\ \mathtt{n}$	Downcast
		$\mathtt{e}\ \mathsf{isInstanceOf}\ \mathtt{n}$	Is Instance Of
		$\mathtt{e}\ \mathsf{isOfKind}\ \mathtt{n}$	Is Kind Of
		$\mathsf{Set}\{\overline{\mathtt{e}}\} \mid \mathsf{Bag}\{\overline{\mathtt{e}}\}$	Collection expression
		$\mathtt{e}_1 \rightarrow \mathsf{iterate}(x; y = \mathtt{e}_2 \mid \mathtt{e}_3)$	Collection iteration

We assume a set of variables ranged over by $x, y, \ldots$, disjoint with $\mathbb{R}$, the set of role names. Keyword self is a special variable that refers to the current contextual instance. Built-in operations on primitive types and collection types are ranged over by κ. Moreover, for collection iterations, we only consider the most fundamental and generic iterate operation, of which all other loop operations specified in OCL can be described as a special case.

4.2 Expression evaluation

The evaluation of a constraint $\mathtt{c} = (\mathtt{n}, \mathtt{e})$ in a model M for a contextual instance n, written $\mathsf{eval}(\mathtt{c}, \mathsf{M}, \mathsf{n})$, is well-formed if $\mathtt{c}$ is defined for the metamodel of M, n is an instance of M, and the type of n is a subtype of the context metaclass $\mathtt{n}$ of $\mathtt{c}$. Only well-formed constraint evaluation has a meaning, which amounts to evaluate the boolean expression $\mathtt{e}$ of $\mathtt{c}$ within the model M and an evaluation environment Λ where the keyword self is bound to the concerned contextual

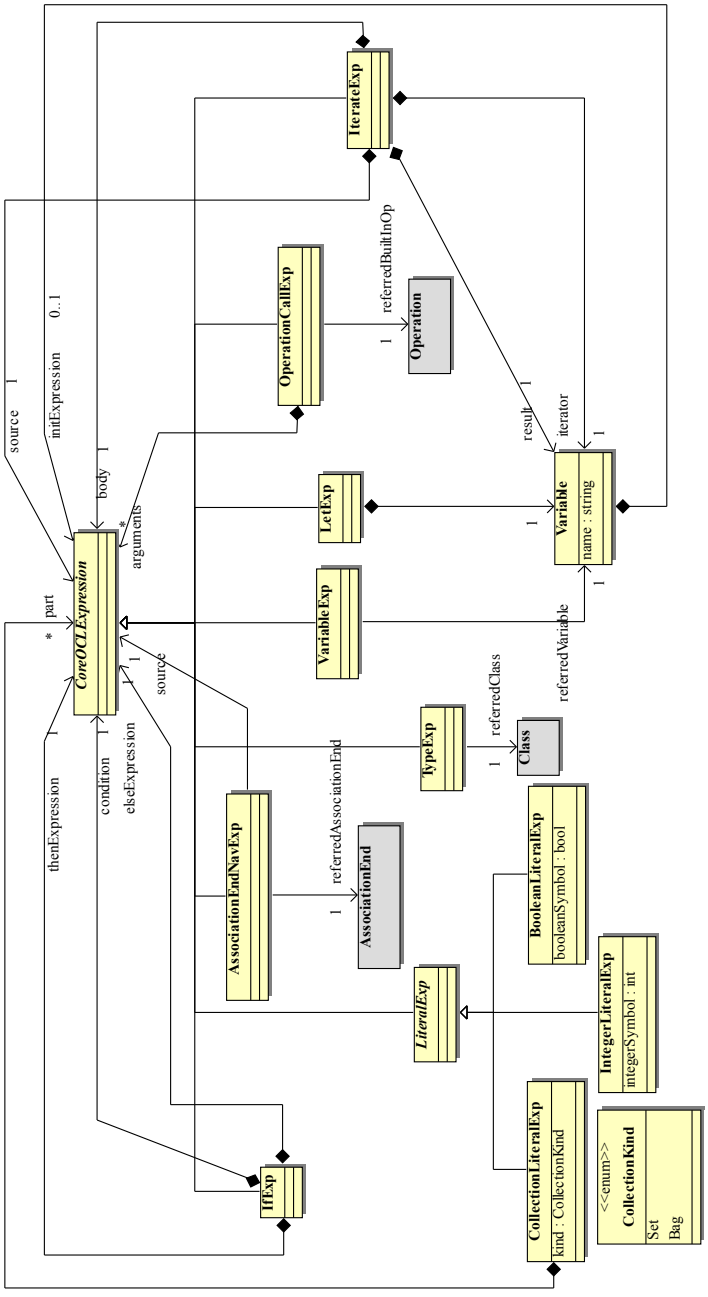

Figure 6: CoreOCL expressions: abstract syntax

instance n. The result consists of three parts: $\mathbb{b}, \mathsf{N}_S, \mathsf{A}_S$, where $\mathbb{b}$ is a truth value (i.e. true or false), and N_S and A_S collect respectively the instances and links of M referenced by $\mathbb{e}$ during the evaluation. We have $\mathsf{eval}(\mathbb{c}, \mathsf{M}, \mathsf{n}) = \mathbb{b}$, and N_S and A_S constitute the sub-model of the scope of $\mathsf{eval}(\mathbb{c}, \mathsf{M}, \mathsf{n})$.

In general, the evaluation of a CoreOCL expression takes the following form of judgment:

$$\Lambda \overset{\mathsf{M}}{\vDash} \mathbb{e} \Downarrow \mathsf{v}, \mathsf{N}_S, \mathsf{A}_S$$

Because CoreOCL expressions also include non-boolean ones, the value v goes beyong truth values as defined below:

v	::=	$\mathbb{i}$	Integer value
		$\mathbb{b}$	Boolean value
		n	Model instance
		null	Null value
		$\mathsf{Set}\{\overline{\mathsf{v}}\} \mid \mathsf{Bag}\{\overline{\mathsf{v}}\}$	Collection value

In addition, we need also the evaluation environment Λ to bind values to any free variables in the expression, as defined below:

$$\Lambda \quad ::= \quad \emptyset \mid (x : \mathsf{v}), \Lambda$$

The context model M is exploited during the course of evaluation for relevant model information such as the types of instances and how instances are linked. The latter is necessary for computing association end navigations. The navigation from an instance n towards an association end via the corresponding role name $\mathbb{r}$, written $\mathsf{n}.\mathbb{r}$ is well-formed if and only if $\tau(\mathsf{n}) \leq \mathsf{fst}(\mathsf{fst}(\mathsf{asso}(\mathbb{r})))$. The result of a well-formed navigation $\mathsf{n}.\mathbb{r}$ in the context of a model M, is either a set, an instance, or null, depending on the target multiplicity of $\mathsf{asso}(\mathbb{r})$. More specifically, we have the following definition:

Definition 8 (Computation of association end navigation)**.** Given a well-formed association end navigation $\mathsf{n}.\mathbb{r}$, it is computed following the rules below:

1. if $\mathsf{snd}(\mathsf{snd}(\mathsf{snd}(\mathsf{asso}(\mathbb{r})))) > 1$, namely the target multiplicity of $\mathsf{asso}(\mathbb{r})$ has a larger-than-one upper bound, $\mathsf{n}.\mathbb{r} \overset{\mathsf{M}}{=} \mathsf{Set}\{\mathsf{n}_1, \ldots, \mathsf{n}_k\}$, where for all $1 \leq i \leq k$, $(\mathsf{n}, (\mathsf{n}_i, \mathbb{r})) \in \mathsf{A}$;
2. if $\mathsf{snd}(\mathsf{snd}(\mathsf{snd}(\mathsf{asso}(\mathbb{r})))) \leq 1$, namely the target multiplicity of $\mathsf{asso}(\mathbb{r})$ has an at-most-one upper bound,

$$\mathsf{n}.\mathbb{r} \overset{\mathsf{M}}{=} \begin{cases} \mathsf{n}' & \exists \mathsf{n}' \text{ such that } (\mathsf{n}, (\mathsf{n}', \mathbb{r})) \in \mathsf{A} \\ \mathsf{null} & \text{otherwise} \end{cases}$$

We define in Figure 7 and 8 the evaluation rules of CoreOCL expressions. Wherever the context model information is required for a computation, M appears explicitly on top of the corresponding computation sign. We say an evaluation judgment holds if there exists an evaluation derivation tree for it, which is constructed by applying the evaluation rules.

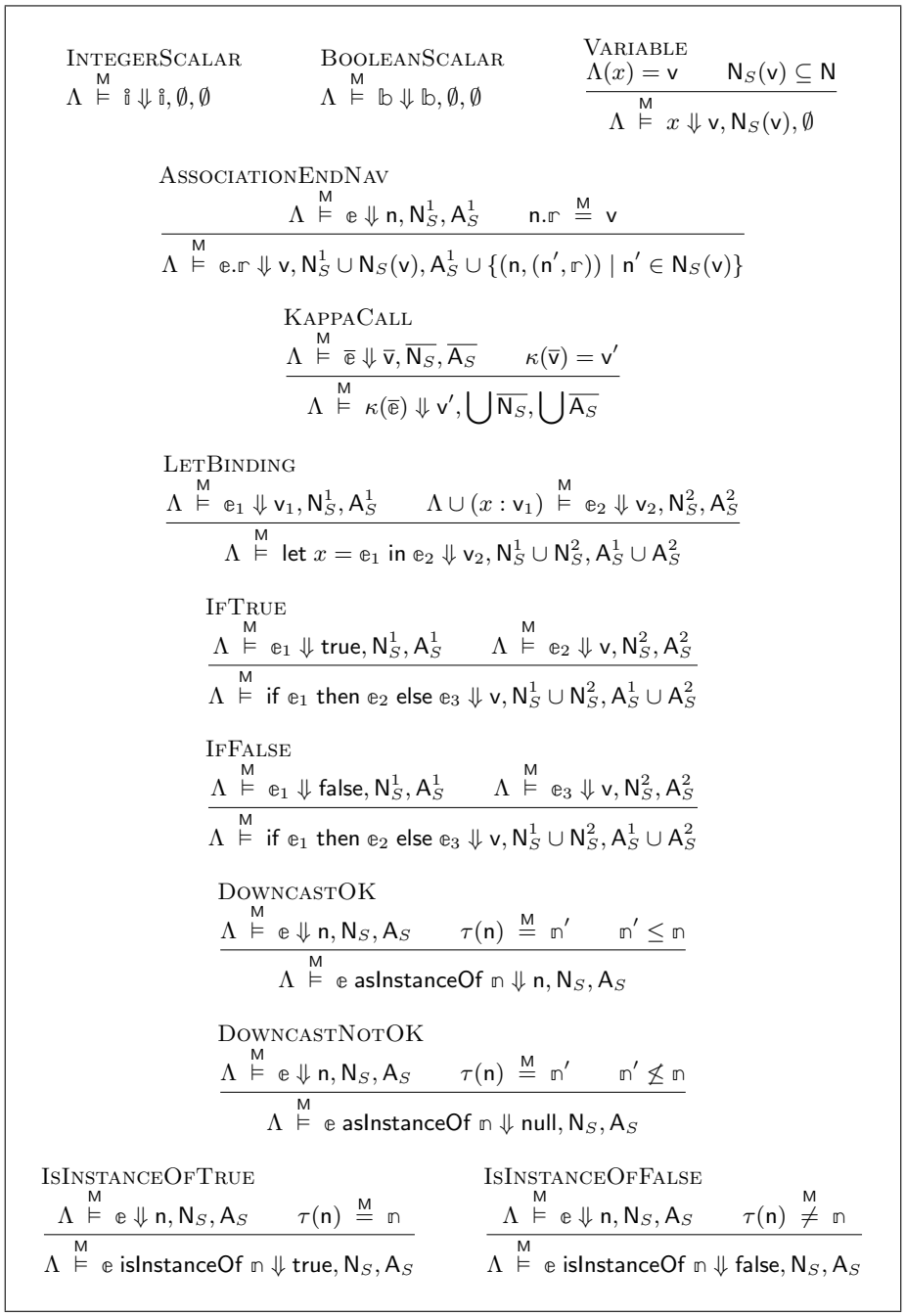

Figure 7: Evaluation rules for expressions: part 1

IsKindOfTrue

$$\frac{\Lambda \overset{\mathsf{M}}{\vDash} \mathbb{e} \Downarrow \mathsf{n}, \mathsf{N}_S, \mathsf{A}_S \qquad \tau(\mathsf{n}) \overset{\mathsf{M}}{=} \mathbb{m}' \qquad \exists \mathbb{m}_1, \ldots, \mathbb{m}_k, \text{ such that } \mathbb{m}' = \mathbb{m}_1 \text{ and } \mathbb{m}_k = \mathbb{m} \text{ and } (\mathbb{m}_i, \mathbb{m}_{i+1}) \in \mathbb{H}, i = 1, \ldots, k-1}{\Lambda \overset{\mathsf{M}}{\vDash} \mathbb{e} \ \mathsf{isOfKind}\ \mathbb{m} \Downarrow \mathsf{true}, \mathsf{N}_S, \mathsf{A}_S}$$

IsKindOfFalse

$$\frac{\Lambda \overset{\mathsf{M}}{\vDash} \mathbb{e} \Downarrow \mathsf{n}, \mathsf{N}_S, \mathsf{A}_S \qquad \tau(\mathsf{n}) \overset{\mathsf{M}}{=} \mathbb{m}' \qquad \nexists \mathbb{m}_1, \ldots, \mathbb{m}_k, \text{ such that } \mathbb{m}' = \mathbb{m}_1 \text{ and } \mathbb{m}_k = \mathbb{m} \text{ and } (\mathbb{m}_i, \mathbb{m}_{i+1}) \in \mathbb{H}, i = 1, \ldots, k-1}{\Lambda \overset{\mathsf{M}}{\vDash} \mathbb{e} \ \mathsf{isOfKind}\ \mathbb{m} \Downarrow \mathsf{false}, \mathsf{N}_S, \mathsf{A}_S}$$

SetExp

$$\frac{\Lambda \overset{\mathsf{M}}{\vDash} \overline{\mathbb{e}} \Downarrow \overline{\mathsf{v}}, \overline{\mathsf{N}_S}, \overline{\mathsf{A}_S}}{\Lambda \overset{\mathsf{M}}{\vDash} \mathsf{Set}\{\overline{\mathbb{e}}\} \Downarrow \mathsf{Set}\{\overline{\mathsf{v}}\}, \bigcup \overline{\mathsf{N}_S}, \bigcup \overline{\mathsf{A}_S}}$$

BagExp

$$\frac{\Lambda \overset{\mathsf{M}}{\vDash} \overline{\mathbb{e}} \Downarrow \overline{\mathsf{v}}, \overline{\mathsf{N}_S}, \overline{\mathsf{A}_S}}{\Lambda \overset{\mathsf{M}}{\vDash} \mathsf{Bag}\{\overline{\mathbb{e}}\} \Downarrow \mathsf{Bag}\{\overline{\mathsf{v}}\}, \bigcup \overline{\mathsf{N}_S}, \bigcup \overline{\mathsf{A}_S}}$$

CollectionIteration

$$\frac{\Lambda \overset{\mathsf{M}}{\vDash} \mathbb{e}_1 \Downarrow \mathit{Collection}\{\mathsf{v}'_1, \ldots, \mathsf{v}'_k\}, \mathsf{N}^0_S, \mathsf{A}^0_S \qquad \Lambda \overset{\mathsf{M}}{\vDash} \mathbb{e}_2 \Downarrow \mathsf{v}_1, \mathsf{N}^1_S, \mathsf{A}^1_S \qquad (\Lambda \cup (x : \mathsf{v}'_i) \cup (y : \mathsf{v}_i)) \overset{\mathsf{M}}{\vDash} \mathbb{e}_3 \Downarrow \mathsf{v}_{i+1}, \mathsf{N}^{i+1}_S, \mathsf{A}^{i+1}_S)^{\,i=1,\ldots,k}}{\Lambda \overset{\mathsf{M}}{\vDash} \mathbb{e}_1 \to \mathsf{iterate}(x; y = \mathbb{e}_2 \,|\, \mathbb{e}_3) \Downarrow \mathsf{v}_{k+1}, \bigcup_{1 \le i \le k+1} \mathsf{N}^i_S, \bigcup_{1 \le i \le k+1} \mathsf{A}^i_S}$$

Figure 8: Evaluation rules for expressions: part 2

Both rule Variable and AssociationEndNav in Figure 7 rely on an auxiliary definition $\mathsf{N}_S(\mathsf{v})$ that computes the model instances referenced by a value v. The computation is defined recursively for different kinds of values:

$$\begin{aligned}
\mathsf{N}_S(\mathbb{i}) &= \emptyset \\
\mathsf{N}_S(\mathbb{b}) &= \emptyset \\
\mathsf{N}_S(\mathsf{n}) &= \{\mathsf{n}\} \\
\mathsf{N}_S(\mathsf{null}) &= \emptyset \\
\mathsf{N}_S(\mathsf{Set}\{\overline{\mathsf{v}}\}) &= \textstyle\bigcup_{\mathsf{v}_i \in \mathsf{v}} \mathsf{N}_S(\mathsf{v}_i) \\
\mathsf{N}_S(\mathsf{Bag}\{\overline{\mathsf{v}}\}) &= \textstyle\bigcup_{\mathsf{v}_i \in \mathsf{v}} \mathsf{N}_S(\mathsf{v}_i)
\end{aligned}$$

The second premise of rule Variable requires that the model instances referenced by value v should all be included in the set of model instances of M i.e. N in order for the value to be meaningful. Moreover, in the first premise of rule AssociationEndNav in Figure 7, which specifies the evaluation of association end navigation expressions $\mathbb{e}.\mathbb{r}$, we require the value of evaluating the sub-expression $\mathbb{e}$ be a model instance n, because applying an association end navigation on integer values $\mathbb{i}$, boolean values $\mathbb{b}$, and the null value null is meaningless. In case the sub-expression $\mathbb{e}$ evaluates to a collection value, because the meaning of applying an association end navigation on a collection value is to

iterate the association end navigation on all the elements of the collection, it will have to be implemented by using the iterate operation explicitly, accroding to the evaluation semantics.

Finally, the evaluation of $\kappa(\overline{\mathsf{v}})$ in rule KappaCall in Figure 7 is defined by the semantics of the built-in operation κ, such as we will have something like: $\mathsf{isDefined}(\mathsf{null}) = \mathsf{false}$, $+(1,2) = 3$, etc. Moreover, we use the abstract $\mathit{Collection}\{\overline{\mathsf{v}}\}$ to stand for either $\mathsf{Set}\{\overline{\mathsf{v}}\}$ or $\mathsf{Bag}\{\overline{\mathsf{v}}\}$.

4.2.1 Some properties of the evaluation semantics

Lemma 1. *If an evaluation judgment* $\Lambda \overset{\mathsf{M}}{\vDash} \mathbb{e} \Downarrow \mathsf{v}, \mathsf{N}_S, \mathsf{A}_S$ *holds, then:*

1. $\mathsf{N}_S(\mathsf{v}) \subseteq \mathsf{N}$;
2. $\mathsf{N}_S \subseteq \mathsf{N}$;
3. $\mathsf{A}_S \subseteq \mathsf{A}$.

Proof. By induction on the structure of expression $\mathbb{e}$. □

Lemma 2. *Given a model* M*, an evaluation environment* Λ*, and an expression of CoreOCL* $\mathbb{e}$*, if* $\Lambda \overset{\mathsf{M}}{\vDash} \mathbb{e} \Downarrow \mathsf{v}, \mathsf{N}_S, \mathsf{A}_S$ *holds, then for any sub-model* M' *of* M *where* $\mathsf{N}_S \subseteq \mathsf{N}'$ *and* $\mathsf{A}_S \subseteq \mathsf{A}'$, $\Lambda \overset{\mathsf{M}'}{\vDash} \mathbb{e} \Downarrow \mathsf{v}, \mathsf{N}_S, \mathsf{A}_S$ *also holds.*

Proof. We prove by in induction on the depth of the structure of the expression $\mathbb{e}$.

Base cases: expressions of depth 1

($\mathbb{e} \equiv \mathbb{i}$) Trivial.

($\mathbb{e} \equiv \mathbb{b}$) Trivial.

($\mathbb{e} \equiv x$) Because of the same evaluation environment Λ and $\mathsf{N}_S \subseteq \mathsf{N}'$.

Induction cases: expressions of depth n We suppose that for any expression of depth $< n$, the lemma holds. We prove that it is also the case for expressions of depth n. We distinguish the top most structure the expression may have:

($\mathbb{e} \equiv \mathbb{e}_1.\mathbb{r}$) Assume $\Lambda \overset{\mathsf{M}}{\vDash} \mathbb{e}_1.\mathbb{r} \Downarrow \mathsf{v}, \mathsf{N}_S, \mathsf{A}_S$ holds. Following AssociationEndNav, we have $\Lambda \overset{\mathsf{M}}{\vDash} \mathbb{e}_1 \Downarrow \mathsf{n}, \mathsf{N}^1_S, \mathsf{A}^1_S$ holds, $\mathsf{n}.\mathbb{r} \overset{\mathsf{M}}{=} \mathsf{v}$, $\mathsf{N}_S = \mathsf{N}^1_S \cup \mathsf{N}_S(\mathsf{v})$, and $\mathsf{A}_S = \mathsf{A}^1_S \cup \{(\mathsf{n}, (\mathsf{n}', \mathbb{r})) \mid \mathsf{n}' \in \mathsf{N}_S(\mathsf{v})\}$. By induction hypothesis, we have also $\Lambda \overset{\mathsf{M}'}{\vDash} \mathbb{e}_1 \Downarrow \mathsf{n}, \mathsf{N}^1_S, \mathsf{A}^1_S$ holds. According to Lemma 1, $\mathsf{n} \in \mathsf{N}'$ i.e. $\mathsf{n}.\mathbb{r}$ is meaningful in M'. According to Definition 8, the value computed by the association end navigation $\mathsf{n}.\mathbb{r}$ in a model equals the set of instances n' in

that model where there is a link from n to n' and the link is of type $\mathsf{asso}(\mathbb{r})$. It is not difficult to see that $\mathsf{n}.\mathbb{r}$ computes to the same value v in both M and M' because M' is a sub-model of M and all the relevant n''s and the corresponding links from n to them in M are included in the evalution scope N_S and A_S, which are in turn included in M' following the lemma hypothesis. As a consequence, by applying rule AssociationEndNav, we have $\Lambda \overset{\mathsf{M}'}{\vDash} \mathbb{e}.\mathbb{r} \Downarrow \mathsf{v}, \mathsf{N}^1_S \cup \mathsf{N}_S(\mathsf{v}), \mathsf{A}^1_S \cup \{(\mathsf{n}, (\mathsf{n}', \mathbb{r})) \mid \mathsf{n}' \in \mathsf{N}_S(\mathsf{v})\}$ holds, i.e. $\Lambda \overset{\mathsf{M}'}{\vDash} \mathbb{e}_1.\mathbb{r} \Downarrow \mathsf{v}, \mathsf{N}_S, \mathsf{A}_S$ holds.

($\mathbb{e} \equiv \kappa(\mathbb{e}_1, \ldots, \mathbb{e}_k)$) Following rule KappaCall and by induction on the sub-expressions $\mathbb{e}_i$, $1 \leq i \leq k$.

($\mathbb{e} \equiv \mathsf{let}\ x = \mathbb{e}_1\ \mathsf{in}\ \mathbb{e}_2$) Following LetBinding and by induction on the two sub-expressions $\mathbb{e}_1$ and $\mathbb{e}_2$.

($\mathbb{e} \equiv \mathsf{if}\ \mathbb{e}_1\ \mathsf{then}\ \mathbb{e}_2\ \mathsf{else}\ \mathbb{e}_3$) Following IfTrue (resp. IfFalse) and by induction on the sub-expressions $\mathbb{e}_1$ and $\mathbb{e}_2$ (resp. $\mathbb{e}_3$).

($\mathbb{e} \equiv \mathbb{e}\ \mathsf{asInstanceOf}\ \mathbb{n}$) Following DownCastOK (resp. DownCastNotOK), by induction on the sub-expression $\mathbb{e}$, and because M' is a sub-model of M.

($\mathbb{e} \equiv \mathbb{e}\ \mathsf{isInstanceOf}\ \mathbb{n}$) Following IsInstanceOfTrue (resp. IsInstanceOfFalse), by induction on the sub-expression $\mathbb{e}$, and because M' is a sub-model of M.

($\mathbb{e} \equiv \mathbb{e}\ \mathsf{isOfKind}\ \mathbb{n}$) Following IsKindOfTrue (resp. IsKindOfFalse), by induction on the sub-expression $\mathbb{e}$, and because M' is a sub-model of M.

($\mathbb{e} \equiv \mathsf{Set}\{\overline{\mathbb{e}}\}$) Following SetExp and by induction on the sub-expressions in $\overline{\mathbb{e}}$.

($\mathbb{e} \equiv \mathsf{Bag}\{\overline{\mathbb{e}}\}$) Similar to the case above for set expressions.

($\mathbb{e} \equiv \mathbb{e}_1 \rightarrow \mathsf{iterate}(x; y = \mathbb{e}_2 \mid \mathbb{e}_3)$) Following CollectionIteration and by induction on the sub-expressions $\mathbb{e}_1$, $\mathbb{e}_2$ and $\mathbb{e}_3$.

□

Proof of Property 1 for CoreOCL. The result follows as a corollary of Lemma 2. □

4.3 CoreOCL constraints are forward

Lemma 3. *All CoreOCL constraints are forward following Definition 7.*

Proof. Give a model $\mathsf{M} = (\mathbb{M}, \mathsf{N}, \mathsf{A}, \tau)$, an instance $\mathsf{n} \in \mathsf{N}$, a CoreOCL expression $\mathbb{e}$, and an evaluation environment Λ binding all the free variables in $\mathbb{e}$ to values that only refer to instances reachable from n if any, i.e. $\forall x$ that is bound in Λ, $\mathsf{N}_S(\Lambda(x))$ reachable from n. Suppose $\Lambda \overset{\mathsf{M}}{\vDash} \mathbb{e} \Downarrow \mathsf{v}, \mathsf{N}_S, \mathsf{A}_S$. We prove by induction on the depth of the structure of the expression $\mathbb{e}$ the following two statements:

1. N_S and A_S are all reachable from n;
2. $\mathsf{N}_S(\mathsf{v})$ all reachable from n.

As a consequence, this lemma then holds as a corollary.

Base cases: expressions of depth 1

($\mathbb{e} \equiv \mathbb{i}$) Trivial.

($\mathbb{e} \equiv \mathbb{b}$) Trivial.

($\mathbb{e} \equiv x$) Because all values bound in Λ refer to instances reachable from n if any.

Induction cases: expressions of depth n We suppose that for any expression of depth $< n$, the two statements hold. We prove that it is also the case for expressions of depth n. We distinguish the top most structure the expression may have:

($\mathbb{e} \equiv \mathbb{e}_1.\mathbb{r}$) Assume $\Lambda \overset{\mathsf{M}}{\vDash} \mathbb{e}_1.\mathbb{r} \Downarrow \mathsf{v}, \mathsf{N}_S, \mathsf{A}_S$ holds. Following AssociationEndNav, we have $\Lambda \overset{\mathsf{M}}{\vDash} \mathbb{e}_1 \Downarrow \mathsf{n}_1, \mathsf{N}^1_S, \mathsf{A}^1_S$ holds, $\mathsf{n}_1.\mathbb{r} \overset{\mathsf{M}}{=} \mathsf{v}$, $\mathsf{N}_S = \mathsf{N}^1_S \cup \mathsf{N}_S(\mathsf{v})$, and $\mathsf{A}_S = \mathsf{A}^1_S \cup \{(\mathsf{n}_1, (\mathsf{n}', \mathbb{r})) \mid \mathsf{n}' \in \mathsf{N}_S(\mathsf{v})\}$. By induction hypothesis, we have n_1 reachable from n. Therefore, following Definition 8, we have also $\mathsf{N}_S(\mathsf{v})$ reachable from n. Moreover, N_S is reachable from n because both N^1_S (by induction hypothesis) and $\mathsf{N}_S(\mathsf{v})$ are. Finally, A_S is reachable from n because both A^1_S and n_1 are.

($\mathbb{e} \equiv \kappa(\mathbb{e}_1, \ldots, \mathbb{e}_k)$) Assume $\Lambda \overset{\mathsf{M}}{\vDash} \kappa(\mathbb{e}_1, \ldots, \mathbb{e}_k) \Downarrow \mathsf{v}, \mathsf{N}_S, \mathsf{A}_S$ holds. Following KappaCall, we have $\Lambda \overset{\mathsf{M}}{\vDash} \mathbb{e}_i \Downarrow \mathsf{v}_i, \mathsf{N}^i_S, \mathsf{A}^i_S$ holds for $1 \leq i \leq k$, $\kappa(\mathsf{v}_1, \ldots, \mathsf{v}_k) = \mathsf{v}$, $\mathsf{N}_S = \bigcup_{1 \leq i \leq k} \mathsf{N}^i_S$ and $\mathsf{A}_S = \bigcup_{1 \leq i \leq k} \mathsf{A}^i_S$. By induction hypothesis, we have N^i_S, A^i_S and $\mathsf{N}_S(\mathsf{v}_i)$ reachable from n for $1 \leq i \leq k$. Therefore, we have N_S and A_S reachable from n. Moreover, κ being the built-in operations on primitive types and collection types, does not introduce any new instances other than those are referenced in its operands, i.e. v_i for $1 \leq i \leq k$. Hence we have $\mathsf{N}_S(\mathsf{v})$ reachable from n as well.

($\mathbb{e} \equiv \mathsf{let}\ x = \mathbb{e}_1\ \mathsf{in}\ \mathbb{e}_2$) Following LetBinding, and by induction on the two sub-expressions $\mathbb{e}_1$ and $\mathbb{e}_2$.

($\mathbb{e} \equiv \mathsf{if}\ \mathbb{e}_1\ \mathsf{then}\ \mathbb{e}_2\ \mathsf{else}\ \mathbb{e}_3$) Following IfTrue (resp. IfFalse) and by induction on the sub-expressions $\mathbb{e}_1$ and $\mathbb{e}_2$ (resp. $\mathbb{e}_3$).

($\mathbb{e} \equiv \mathbb{e}\ \mathsf{asInstanceOf}\ \mathbb{n}$) Following DownCastOK (resp. DownCastNotOK) and by induction on the sub-expression $\mathbb{e}$.

($\mathbb{e} \equiv \mathbb{e}\ \mathsf{isInstanceOf}\ \mathbb{n}$) Following IsInstanceOfTrue (resp. IsInstanceOfFalse) and by induction on the sub-expression $\mathbb{e}$.

($\mathbb{e} \equiv \mathbb{e}$ isOfKind $\mathbb{n}$) Following IsKindOfTrue (resp. IsKindOfFalse) and by induction on the sub-expression $\mathbb{e}$.

($\mathbb{e} \equiv \mathsf{Set}\{\overline{\mathbb{e}}\}$) Following SetExp and by induction on the sub-expressions in $\overline{\mathbb{e}}$.

($\mathbb{e} \equiv \mathsf{Bag}\{\overline{\mathbb{e}}\}$) Similar to the case above for set expressions.

($\mathbb{e} \equiv \mathbb{e}_1 \rightarrow \mathsf{iterate}(x; y = \mathbb{e}_2 \mid \mathbb{e}_3)$) Following CollectionIteration and by induction on the sub-expressions $\mathbb{e}_1$, $\mathbb{e}_2$ and $\mathbb{e}_3$.

□

5 Application One: Pruning based Model Comprehension

In this section, we demonstrate the power of our generic model decomposition technique by reporting one of its applications in a pruning-based model comprehension method. A typical comprehension question one would like to have answered for a large model is:

> *Given a set of instances of interest in the model, how does one construct a substantially smaller sub-model that is relevant for the comprehension of these instances?*

Model readers, when confronted with such a problem, would typically start from the interesting instances and browse through the whole model attempting to manually identify the relevant parts. Even with the best model documentation and the support of model browsing tools, such a task may still be too complicated to solve by hand, especially when the complexity of the original model is high. Moreover, guaranteeing by construction that the identified parts (together with the interesting instances) indeed constitute a valid model further complicates the problem.

Our model decomposition technique can be exploited to provide a linear time automated solution to the problem above. The idea is to simply take the union of all the scc-nodes, each of which contains at least one interesting instance, and their descendant scc-nodes in the decomposition hierarchy of the original model. We have implemented the idea in an Ecore [9] model comprehension tool [1] based on the implementation of the model decomposition technique.

To assess the applicability of the tool, a case study has been carried out. We have chosen the Ecore model of BPMN (Business Process Modeling Notation) [11] bpmn.ecore as an example, and one of BPMN's main concepts – *Gateway* – for comprehension. Gateways are modeling elements in BPMN used to control how sequence flows interact as they converge and diverge within a business process. Five types of gateways are identified in order to cater to different types of sequence flow control semantics: exclusive, inclusive, parallel, complex, and event-based.

Inputs to the comprehension tool for the case study are the following:

Figure 9: Bird's Eye View of the Class diagram of the BPMN Ecore model

- The BPMN Ecore model containing 134 classes (EClass instances), 252 properties (EReference instances), and 220 attributes (EAttribute instances). Altogether, it results in a very large class diagram that does not fit on a single page if one wants to be able to read the contents properly. Figure 9 shows the bird's eye view of this huge diagram.

- a set of interesting instances capturing the key notions of the design of gateways in BPMN: Gateway, ExclusiveGateway, InclusiveGateway, ParallelGateway, ComplexGateway, and EventBasedGateway.

After applying the tool, the result BPMN sub-model that contains all the selected interesting instances has only 17 classes, 7 properties, and 21 attributes. We observe that all the other independent concepts of BPMN such as *Activity*, *Event*, *Connector*, and *Artifact*, are pruned out. The class diagram view of the pruned BPMN model is shown in Figure 10. Note that it corresponds well to the class diagram that is sketched in the chapter for describing gateways in the BPMN 2.0 specification [11]. We have also verified that the pruned BPMN model is indeed an Ecore instance by validating it against ecore.ecore in EMF [9].

6 Application Two: Hierarchical Full Model Comprehension

Our model decomposition technique can also be used for full model comprehension. Moody has coined in his design theory entitled "The Physics of Notations" [6] a "principle of complexity management" that advocates two mechanisms for effectively dealing with complexity in visual representations: modularization (subsystems) and hierarchy (levels of abstraction). Both mechanisms are available when using our model decomposition technique.

Firstly, our model decomposition technique guides you through the multitude of model elements by using the sub-model lattice (see page 13 for the definition) as the road map. A model reader can understand the sub-models (subsystems) indicated in the lattice one by one. These sub-models can be understood in isolation because they are all valid models conforming to the same

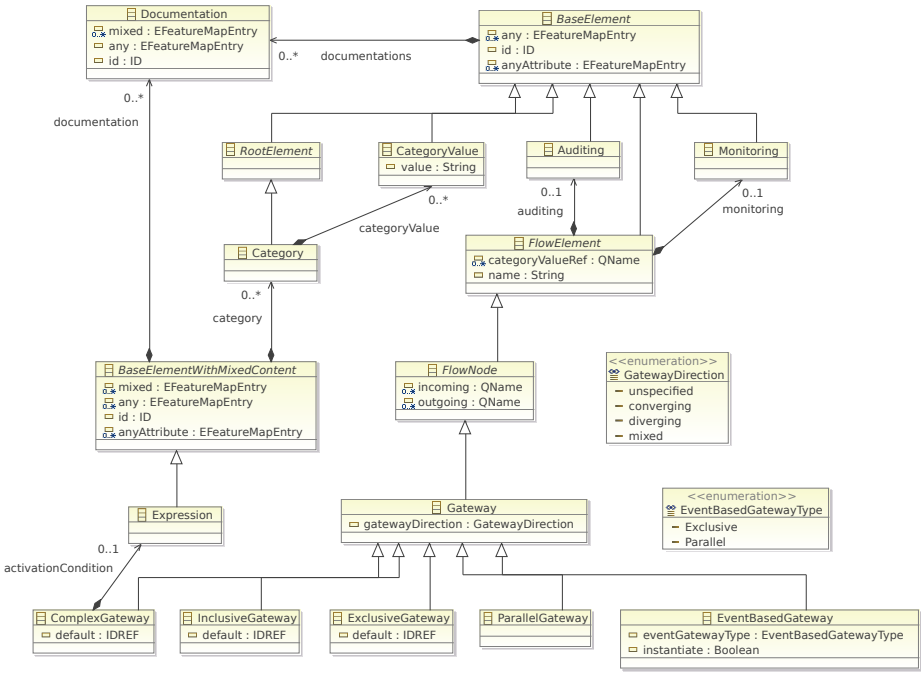

Figure 10: Pruned class diagram for understanding the Gateway concept in BPMN.

metamodel as the original system model hence carrying a well specified semantics (if one is defined). One can choose to start from any level of granularity (i.e. a sub-model with the complexity) that is appropriate for her/his cognitive capability.

Secondly, once a sub-model is understood, the instances and links in it can be abstracted away by folding the whole sub-model up into a single node which is actually an antichain-node (see page 13 for definition) in the decomposition hierarchy of the system model. Such a process of abstraction can be recursively followed and a lattice of the original system model at different levels of abstraction can be built up. This lattice, referred to as *the abstraction lattice*, is isomorphic to the sub-model lattice of the system model, (See page 13 for what a sub-model lattice is.) where the indicated sub-model in the sub-model lattice is abstracted away into the corresponding antichain-node in the abstraction lattice. Moreover, the opposite direction of abstraction, namely refinement, can also be realized by opening up antichain-nodes to view the contained model instances, links, and other possibly nested antichain-nodes. Figure 11 shows the abstraction/refinement lattice of the example system model which is given in Figure 1.

So far an antichain-node is denoted by the set of scc-nodes (see page 9 for

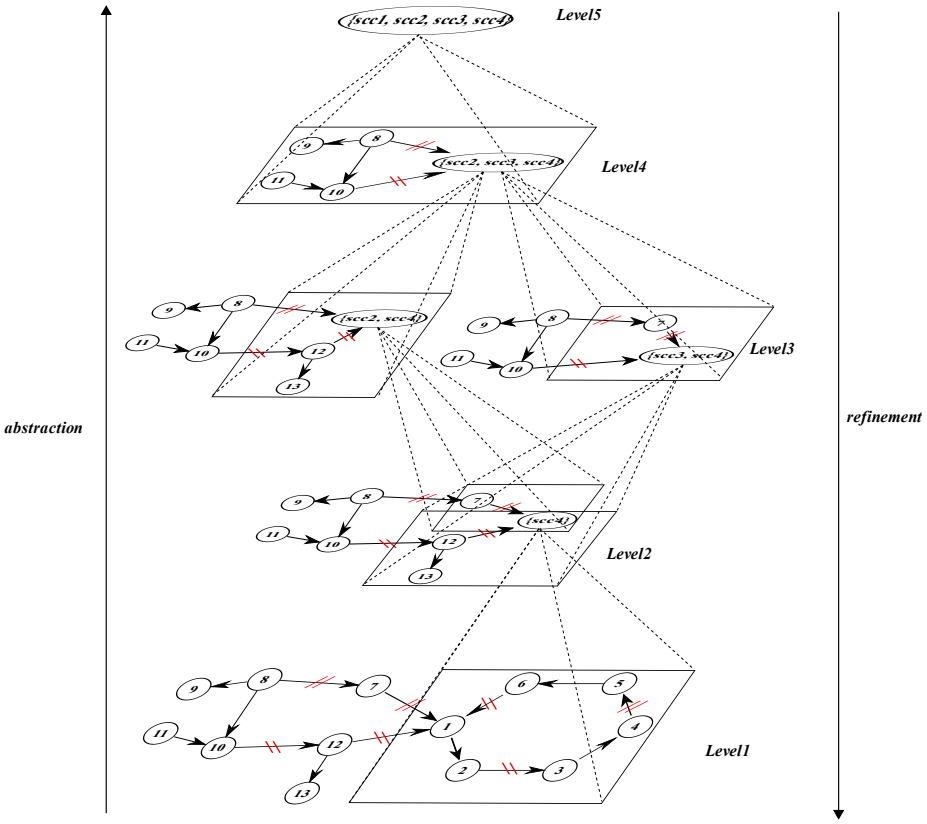

Figure 11: The example model in Figure 1 is represented at different levels of abstraction.

definition) that are collapsed in it, which is not very informative. We propose a mechanism for generating thumbnails of antichain-nodes. Such a thumbnail, which can be used to represent an antichain-node when it is folded up, is meant to help the readers to have a rough overview of the internal organization of the antichain-node. Thumbnails have substantially reduced complexity in comparison to the original set of instances and links that are collapsed in the antichain-node. More specifically, the thumbnail of an antichain-node is constructed by the so called "representative" model elements among those collapsed in it.

Definition 9 (Antichain-node thumbnail)**.** For an antichain-node in an decomposition hierarchy, the thumbnail of it is constructed by the set of representative model elements in it selected recursively as follows:

1. the root instances, i.e. those without incoming internal links are representative;

2. instances and links on root circles are representative; a root circle is a cycle where removing the links on the circle makes all the instances on the circle root instances;

3. all the source and target instances of external links are representative;

4. the parent instance that points to a representative instance via a composition link is also representative, so is the composition link itself [1].

6.1 Tooling

We have implemented a tool following the ideas discussed above for hierarchical full model comprehension based on the implementation of the model decomposition technique. Briefly speaking, it implements the following functionalities:

- View a sub-model satisfying both Condition 1 and 2 for comprehension in separation. Such a sub-model is a valid model conforming to the same metamodel as the original system model (assuming all constraints are forward constraints), hence can be understood independently.
- Abstract a sub-model into an antichain-node.
- Compute the thumbnail of an antichain-node which summarizes the internal structure of the model elements collapsed in the antichain-node.
- Refine an antichain-node into a sub-model if the model reader wants to view again a previously viewed model.

A typical scenario of how a model reader uses this tool for full model comprehension is described below:

1. The model reader chooses a model for comprehension.
2. The tool displays the model as a graph. This is the initial view of the model in the tool.
3. Based on the current view of the model, the tool offers a list of aggregations of elements from the current view, which are either original model elements or antichain-nodes, accompanied with their approximate complexity in terms of numbers of original model elements that are included (for antichain-nodes we will count the number of original model elements that are collapsed in them) for the model reader to comprehend in separation. Note that each such aggregation denotes a sub-model in the sub-model lattice. The list is ordered according to increasing complexity.

[1] Composition associations are not distinguished in our metamodel definition but can be easily patched. Links of composition associations are not fragmentable. Although composition associations normally have a "0..1" source cardinality, the actual lower bound is not unconditionally 0. An additional constraint is often involved, saying that the total number of instances that targets the current instance via a composition link should always adds up to 1. This is an invariant that needs the power of "allInstances" to specify.

4. The model reader chooses from the list an aggregation for comprehension.
5. The tool displays the aggregation as a graph.
6. The model reader reads and comprehends the aggregation, by navigating the graph, or/and opening nested antichain-nodes and folding up again.
7. The model reader finishes understanding the aggregation and requires to fold it up into an antichain-node.
8. The tool computes the thumbnail for the antichain-node.
9. The tool positions and highlights the antichain-node in the graph of the overall model, in order to give the model reader a flavor about how much of the model and where in the model he has comprehended. A new view of the model is derived.

   (Begin alternative steps 6-9)
10. The model reader cancels understanding the aggregation. The view of the model remains the same.

    (End alternative steps 6-9)
11. Goto step 3 until the whole model is understood.

7 Conclusion

The lattice of sub-models described in this paper should have applications beyond the applications for model comprehension described in this paper. We foresee potential applications in the areas of model testing, debugging, and model reuse.

Given a software that takes models of a metamodel as input (e.g., a model transformation), an important part in testing the software is test case generation. Our model decomposition technique could help with the generation of new test cases by using one existing test case as the seed. New test cases of various complexity degree could be automatically generated following the sub-model lattice of the seed test case.

Moreover, our model decomposition can also help with the debugging activity when a failure of the software is observed on a test case. The idea is that we will find a sub-model of the original test case which is responsible for triggering the bug. Although both the reduced test case and the original one are relevant, the smaller test case is easier to understand and investigate.

A major obstacle to the massive model reuse in model-based software engineering is the cost of building a repository of reusable model components. A more effective alternative to creating those reusable model components from strach is to discover them from existing system models. Sub-models of a system extracted by following our model decomposition technique are all guarenteed to be valid models hence can be wrapped up into modules and reused

in the construction of other systems following our modular model composition paradigm [5].

Our model decomposition technique, described in this paper can be further improved: indeed it is currently based on two sufficient conditions that are not necessary. A consequence of this is that not all conformant sub-models are captured in the lattice of sub-models. A finer analysis of the constraints in the metamodel could result in weakening the two conditions and thus provide a more complete collection of conformant sub-models.

References

[1] Democles tool. `http://democles.lassy.uni.lu/`.

[2] Jung Ho Bae, KwangMin Lee, and Heung Seok Chae. Modularization of the UML metamodel using model slicing. *Fifth International Conference on Information Technology: New Generations*, 0:1253–1254, 2008.

[3] F.K. Frantz. A taxonomy of model abstraction techniques. *Winter Simulation Conference*, 0:1413–1420, 1995.

[4] Huzefa Kagdi, Jonathan I. Maletic, and Andrew Sutton. Context-free slicing of UML class models. In *ICSM '05: Proceedings of the 21st IEEE International Conference on Software Maintenance*, pages 635–638, Washington, DC, USA, 2005. IEEE Computer Society.

[5] Pierre Kelsen and Qin Ma. A modular model composition technique. In *the Proceedings of13th International Conference on Fundamental Approaches to Software Engineering, (FASE 2010)*, volume LNCS 56013, pages 173–187, 2010.

[6] Daniel Moody. The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering*, 35(6):756–779, 2009.

[7] OMG. Object Constraint Language version 2.2, February 2010.

[8] Sagar Sen, Naouel Moha, Benoit Baudry, and Jean-Marc Jézéquel. Metamodel pruning. In *MoDELS*, pages 32–46, 2009.

[9] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework, 2nd Edition.* Addison-Wesley Professional, 2008.

[10] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[11] Stephen A. White and Derek Miers. *BPMN Modeling and Reference Guide.* Future Strategies Inc., 2008.