A Modular Model Composition Technique

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

TR-LASSY-09-01

Abstract

Model composition is a technique for building bigger models from smaller models, thus allowing system designers to control the complexity of a model-driven design process. Model composition is usually a two-step process: a first matching step identifies corresponding elements in the participating models; a following merge step defines how these corresponding elements are to be fused in the composed model. These model composition techniques may be called white-box model composition since they assume full access to the elements in the participating models.

In this paper we present a model composition technique with a more modular flavor that treats the participating models as black boxes. Our technique has several desirable features: it is applicable to arbitrary metamodels; it is formally defined; the resulting composed model is easy to understand because of the modular nature of the model composition. We demonstrate the use of our model composition technique by presenting two applications, one to executable modeling and the other one to model comprehension.

1 Introduction

Models are the primary artifacts in a model-driven software development process. Models help in dealing with the complexity of the underlying domains by abstracting away irrelevant details. The models themselves can become quite large, at least if we try to represent complex problem or solution domains. Thus we need techniques for tackling model complexity.

One such technique is model composition. By composing large models from smaller models using composition the large models become easier to understand and to maintain. Several model composition techniques have been proposed. For instance in the area of aspect-oriented modeling multiple models are used to represent different facets of a system. Model composition techniques are then used to combine the individual models representing the different views into a model of the overall system.

Most current model composition techniques assume full access to the elements inside a model. For instance model composition based on weaving describe how an aspect model is woven into a base model at given join points. These model compositions can be quite complex themselves in the sense that they merge the internal elements of the participating models in non-trivial ways.

To reap the full benefits of model composition, one needs to keep the composition mechanisms simple. One way to reduce the complexity of the model composition is information hiding. This has been used successfully at the programming level and was introduced in the seminal paper of David Parnas [12]. The basic idea of information hiding is to separate the code into separate pieces called modules that expose only a small subset of their internal elements to the other modules. Modular programming offers several advantages:

- Manageability: modules are smaller in size, easier to read and understand.
- Efficiency: modules can be developed in parallel by different teams reducing the overall system development time; teams with appropriate expertise are assigned to tackle each module increasing the quality of solutions.
- Reusability: modules can be reused within a program or across programs.
- Flexibility: individual modules can be updated or replaced independently as long as the interfaces match, while leaving the other part of the program intact.

In this paper we present a model composition technique that tries to achieve some of the benefits of modular programming at the modeling level. Indeed, the models that participate in the model composition offer an interface that contains a subset of the model's elements; only elements in the interface can be used in the composition. This simplifies the semantics of the composition since the interface allows us to abstract from the internal representation of the models.

More specifically, this work tries to answer the following questions:

- What is modular modeling in general analogous to modular programming?
- How can we enforce modular modeling even the modeling language does not provide explicit supports for doing so?
- What are modules at modeling level?
- How shall we define the interface of a model?
- How modules are composed together to render the whole system model?
- And finally, how a given legacy model can be decomposed into modules for better comprehension?

The presentation of this paper is as follows: in the next section we present an informal overview of the ideas in the paper; its role is that of a roadmap for the remainder of the paper. The following sections which are of a more technical nature should be easier to understand with the help of this roadmap. In section 3 we introduce formal definitions of models, metamodels and model conformance. We then introduce the notion of fragment metamodels in section 4. In section 5 we equip models with an interface. We then present our model composition technique in section 6. We show that this technique yields hierarchies of models in section 7. These model hierarchies can be used to facilitate model comprehension, as we show in section 8. The final two sections present related work and concluding remarks.

2 Overview

In its most general form model composition takes as input several models - which we term *participant models* - that may conform to different metamodels [3].

We base our work on a simpler case: the participant models have a common metamodel and they are disjoint, that is, they do not share common model elements. This situation is depicted in figure 1 in which the individual blobs at the bottom represent the disjoint participant models.

The basic idea behind our approach is to compose the participant models using *fragments*. Fragments are essentially partial models that have (external) links to other models. In figure 1 the cut-off blob at the top represents the fragment: the external links from the fragment are show as dashed arrows connecting the fragment to the models.

Because we plaqce ourselves fully within a model-driven approach, we would like to view fragments as models, too. This is accomplished in section 4 by extending the metamodel of the participant models into a fragment metamodel. This is done by identifying associations in the metamodel that can be instantiated as external links - we call these associations *fragmentation points* - and by adding classes to the metamodel that represent external connection points - so-called referential nodes. We then define fragments as instances of the fragment metamodel.

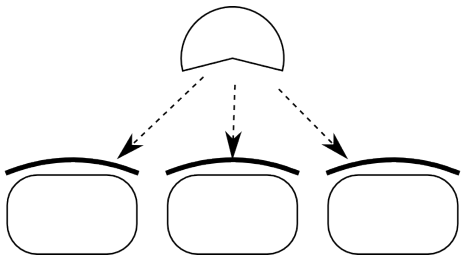


Figure 1: Modular model composition

We implement information hiding by allowing models to offer only a subset of model elements as possible targets of external fragment links: we call this subset of connection points the model interface (see section 5). We represent the model interfaces in figure 1 by curved lines above each model which the external fragment links connect to. In section 6 we then describe the model composition technique that takes as input a set of models, a fragment, and a mapping of external links of the fragments to concrete elements in the participant models. This section also contains the main result of the paper, namely that the result of our model composition technique is a model conforming to the same metamodel as the participant models.

In section 7 we show that this model composition can be applied repeatedly to build a model hierarchy: this is essentially a directed acyclic graph whose nodes are fragments and models. Each fragment node represents the model obtained by composing the models represented by the successor nodes. Thus we may view a model hierarchy as a recipe to construct a complex model by a series of model composition steps.

In section 8 we show that we can reserve this process: instead of starting with simple models and building more complicated models using successive model composition steps, we can start with a complex model and decompose it into a model hierarchy. We explain how such a model decomposition step can be used to facilitate model comprehension.

3 Basic Definitions: Models and Metamodels

Before talking about models, we first need to understand the language in which models can be expressed.

3.1 Metamodels

Following the interpretation of the Object Management Group (OMG) a modeling language is a metamodel: a model of models. A metamodel generalizes and defines the constructs and constraints following which models are built by instantiating the constructs. In short, a metamodel consists of a finite set of classes and a finite set of associations that are basically relations between classes. A graph-based representation of metamodels is employed, where we use nodes to represent metamodel classes and edges metamodel associations. A special kind of relation between classes, called inheritance, is captured separately from other associations, because this relation has no instances in models, but only serves as sub-typing specification: a class B being a sub-type of a class A means that in models, an instance of B can appear in any places where an instance of A is expected.

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

- *N is a set of nodes, representing the set of classes.*
- *$E \subseteq (N \times \mu) \times (N \times \mu)$, where $\mu \subseteq Int \times \{Int \cup \{\infty\}\}$. It represents the set of associations, with the two N's being the types of association ends, and the two μ's being the corresponding multiplicities. We refer to the first end of the edge the source, and the second the target.*
- *$H \subseteq N \times N$ denotes the inheritance relation among classes, where for a given $h \in H$, $\mathsf{fst}(h)$ inherits from (i.e. is a sub-type of) $\mathsf{snd}(h)$.*

3.2 Models

A model in our sense is an abstract representation of a software system from a certain point of view. It captures some of the subject matters of the system while leaving the others out. As a consequence, models need not be complete with respect to the system. In fact, a system is normally captured by many different models and only the composition of them render the whole model of the system. In the modular modeling paradigm, models are naturally considered as modules. The full definition of modules in modular modeling appears later in Section 5.

Models are built by instantiating the constructs, i.e. classes and associations, of a metamodel. Again, a graph-based representation of models can be employed, in which instances of metamodel classes (i.e. metamodel nodes) give rise to nodes in model graphs and instances of metamodel associations (i.e. metamodel edges) render corresponding edges between the model graph nodes. The types of the nodes and edges in model graphs, i.e. from which they are instantiated, are explicitly recorded in terms of a function. For a given model graph node (or edge), the function returns the corresponding metamodel graph node (or edge).

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

- *$\mathcal{M}$ is the metamodel in which the model is expressed.*
- *N is a set of nodes. They are instances of nodes in the metamodel $\mathcal{M}$, i.e. $N_{\mathcal{M}}$.*

- *$E \subseteq N \times N$ is a set of edges. They are instances of edges in the metamodel $\mathcal{M}$, i.e. $E_{\mathcal{M}}$. Edges in models are often referred to as links.*
- *τ is the typing function: $(N \to N_{\mathcal{M}}) \cup (E \to E_{\mathcal{M}})$. It records the type information of the nodes and links in the model, i.e. of which metamodel constructs the nodes and links are instances.*

3.3 Model conformance

Not all models following the definitions above are valid, or said "conforming to" the metamodel. Several conditions need to be fulfilled in addition:

1. Links only relate instances of metamodel classes that are sub-types of the metamodel classes as specified in the corresponding associations of which the links are instances.
2. The number of instances that are related to an instance via links of a metamodel association should fall in the range of the multiplicity specified for that corresponding association ends.

We formalize the conditions by the following well-formedness rules.

Definition 3 (Model conformance)**.** *We say a model $M = (N, E, \mathcal{M}, \tau)$ conforms to its metamodel $\mathcal{M}$ or is well-formed when the following two conditions are met:*

1. *type compatible: $\forall e \in E$, $\tau(\mathsf{fst}(e)) \leq \mathsf{fst}(\mathsf{fst}(\tau(e)))$ [1] and $\tau(\mathsf{snd}(e)) \leq \mathsf{fst}(\mathsf{snd}(\tau(e)))$. Namely, the types of the link ends must be compatible to (being sub-types of) the types as specified in the corresponding association ends.*
2. *multiplicity compatible: $\forall n \in N, e_{\mathcal{M}} \in E_{\mathcal{M}}$,*

   if $\tau(n) \leq \mathsf{fst}(e_{\mathcal{M}})$,

   then $\sharp\{e \mid e \in E$ and $\tau(e) = e_{\mathcal{M}}$ and $\mathsf{fst}(e) = n\} \in \mathsf{snd}(\mathsf{snd}(e_{\mathcal{M}}))$ [2];

   if $\tau(n) \leq \mathsf{snd}(e_{\mathcal{M}})$,

   then $\sharp\{e \mid e \in E$ and $\tau(e) = e_{\mathcal{M}}$ and $\mathsf{snd}(e) = n\} \in \mathsf{snd}(\mathsf{fst}(e_{\mathcal{M}}))$.

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

4 Fragment Metamodels

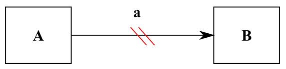
As mentioned above, models usually just capture some subject matter of a system. We need to assemble these different pieces into a coherent whole in order to get the system model. Many approaches have been proposed to support this

[1] $\leq$ denotes the subtyping relation.
[2] $\sharp$ returns the size of a set.

procedure of model assembling, such as aspect weaving or merging based model composition. We consider these approaches a bit too much destructive because the original models usually do not appear directly in the assembled models. This subsequently impairs traceability of unit models and prevents their evolution independently. Moreover, these approaches are also heavy-weighted, in two senses. First, models that are re-used in multiple places in the system are duplicated in the assembled system model. Second, they normally involve the learning of yet another language that is dedicated to specifying the composition policies, ranging from simple template parameter binding sentences, to complicated point-cut designation notations. The comprehension of the result assembled model is thus heavily dependent on the comprehension of the semantics of the composition language.

We propose in this work a light-weighted solution to the problem with a less destructive form of model composition. The language that is used to express the composition models, i.e. the glues that integrate models into a whole, is derived from the metamodel of the models directly, which we call the fragment metamodel.

The fragment metamodel is itself a metamodel and we refer to the models of the fragment metamodel, fragments. Fragments are used to join models of the original metamodel together. Please refer to Section 6 for further details. The fragment metamodel inherits all the constructs and constraints from the original metamodel, and in addition identifies the so called "fragmentation points" in the original metamodel. Following the graph representation of metamodels in Definition 1, a fragmentation point resides on an association edge. As an example, the following diagram specifies a fragmentation point on the association "a" between class "A" and "B", by two parallel short red lines cutting into the association edge.



The semantics of a fragmentation point needs to be understood from two facets. On one hand, a fragmentation point designates a visible spot from the point of view of the original metamodel. More specifically in this example, any instances of class "B" (i.e. the target class of the association) in a model is visible to the external with respect to the "a" type links in the sense that the instance can be hooked upon via an "a" type link from outside the model. From this point of view, the fragmentation points all together define the default interface of a model module. It is the default one because no explicit interface specification is needed, it is defined at the metamodel level, and moreover, the module designer may decide to further hide more information by restricting the visible spots to only a subset of the fragmentation points. Note that a fragmentation point is specified in terms of the association instead of the target class in order to offer fine grained control of visibility where instances of a class can be visible with respect to an association but not another.

On the other hand, a fragmentation point designates a referenceable spot from the point of view of the fragment metamodel. More specifically in the example above, any instances of class "A" (i.e. the source class of the association) in a fragment can connect to a referential "B" instance via an "a" type link. A referential instance of a class is not an instance of the class in normal sense. It semantically represents a place-holder that will be filled up by a normal instance of the class or its sub-classes during module integration. Details are to be discussed in Section 6.

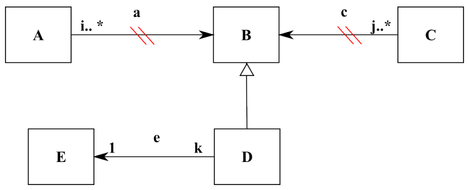
But what after all is a a fragmentation point in a metamodel? Following the discussions above, a fragmentation point syntactically takes the form: a, where a denotes a metamodel association. However, not all metamodel associations give meaningful fragmentation points. In particular, we shall respect the following selection criterion: The multiplicity of the source association end of a must have an infinite upper bound.

Formally, we define the fragmentation points of a metamodel as follows.

Definition 4 (Fragmentation points of a metamodel)**.** *A fragmentation point a of a metamodel $\mathcal{M} = (N, E, H)$ satisfies the following conditions:*

1. *$a \in E$.*
2. *$\mathsf{snd}(\mathsf{fst}(a)) = (\_, \infty)$, where $\_$ represents any integer whose value is irrelevant for this definition.*

Let us consider a concrete example. Given the following metamodel, where the multiplicities are given in the UML style, i.e. with $*$ standing for ∞ and i, j, k representing some integers, the set of fragmentation points are: $\{a, c\}$. Note that the association e is not included into the set of fragmentation points, because its source end multiplicity does not satisfies the condition above.



Based on the definition of fragmentation points in a metamodel, the fragment metamodel is derived by extending the original metamodel with a pair of a referential association edge for each fragmentation point; and a cloned referential class node for the target class of the fragmentation points. The cloned referential class node and the original class node co-exist in the fragment metamodel. Moreover, they two are generalized into a common abstract super class. By contrast, the new referential association edge, which possesses the same source class, the same multiplicities on both association ends, but points to the newly created abstract super class as the target, replaces the original association edge. As a consequence, the fragment metamodel allows both normal instances and referential instances to be connected to the links of the referential associations. A formal definition of the fragment metamodel is given below:

Definition 5 (Fragment metamodel)**.** *The fragment metamodel $\mathcal{M} = (N, E, H)$ is a metamodel, written $\mathcal{M}_F = (N_F, E_F, H_F)$. It is constructed as follows:*

1. *$N \subseteq N_F$, $H \subseteq H_F$.*
2. *$\forall e \in E$,*

   if e is not a fragmentation point of $\mathcal{M}$,

   then $e \in E_F$;

   else (that is the target of the given edge e, i.e. $\mathsf{snd}(e)$ can be "referential"), let $n = \mathsf{snd}(e)$,

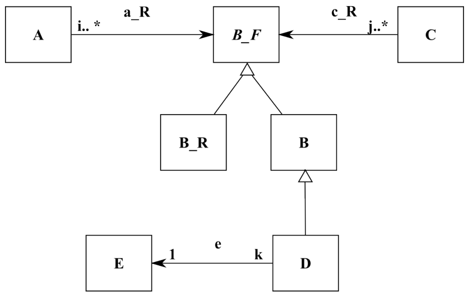
   (a) if n is not yet cloned, i.e. the referential counterpart does not exist yet,

   then create a new node n_R. We call n_R the referential node of n. Moreover, also create a new node n_F (n_F is an abstract node in the sense that no instances can be made from it), and let $(n, n_F) \in H_F$ and $(n_R, n_F) \in H_F$.

   else i.e. n is already cloned, i.e. n_F, n_R exist, then do nothing.

   (b) Create a new edge e_R called the referential edge of e of the following form $(\mathsf{fst}(e), (n_F, \mathsf{snd}(\mathsf{snd}(e))))$, $e_R \in E_F$.

The fragment metamodel of the example above looks like the following according to definition:



Following the definition above, it is not difficult to see that the set of nodes in the fragment metamodel can be divided into three disjoint sub-sets: those inherited from the original metamodel N_F^O, the referential ones N_F^R, and those newly added to act as the common super classes N_F^F. Similarly, the set of edges in the fragment metamodel can also be divided into two disjoint sub-sets: those inherited from the original metamodel E_F^O and the referential ones E_F^R. Moreover, we have the following properties hold.

Property 1. *The mapping from E_F^R to E and from N_F^R to N written $\mathsf{ori}(\cdot)$, is one-to-one. We refer to the reverse mapping of $\mathsf{ori}(\cdot)$ by $\mathsf{ref}(\cdot)$.*

Property 2. *For a given $e_R \in E_F^R$, we have:*

1. *e_R and $\mathsf{ori}(e_R)$ have the same source end:* $\mathsf{fst}(e_R) = \mathsf{fst}(\mathsf{ori}(e_R))$*;*

2. *the target end type of* $\mathsf{ori}(e_R)$ *is a sub-type of the target end type of e_R:* $\mathsf{fst}(\mathsf{snd}(\mathsf{ori}(e_R))) \leq \mathsf{fst}(\mathsf{snd}(e_R))$*;*

3. *e_R and $\mathsf{ori}(e_R)$ have the same target end multiplicity:* $\mathsf{snd}(\mathsf{snd}(e_R)) = \mathsf{snd}(\mathsf{snd}(\mathsf{ori}(e_R)))$.

Property 3. *Given a fragment metamodel, for any referential association $e_R \in E_F^R$, e_R has the same source end as $\mathsf{ori}(e_R)$. Moreover, the multiplicity of the source end has an infinite upper bound, i.e.* $\mathsf{snd}(\mathsf{fst}(e_R)) = \mathsf{snd}(\mathsf{fst}(\mathsf{ori}(e_R))) = (\_, \infty)$.

5 Model Interfaces

We are now ready to define the concept of modules in the modular modeling paradigm. Any metamodel can be enriched with the capability to support modular modeling. Shortly, a module is a model conforming to the metamodel equipped with an interface.

Definition 6 (Modules)**.** *A module is a self-contained model conforming to a metamodel with an interface. It captures a specific subject matter of a system.*

A module interacts with its context via its interface. Module interfaces are specified in terms of a set of pairs of form $(a, i : C)$, where C is the name of a metamodel class, i denotes an instance of C in the module, and a is the name of a metamodel association of which C is (a sub-class of) the target class. Note that i is optional. In case of absence, any instances in the module that are of type C is considered.

Without explicit specification, each module, a model conforming to a metamodel, is equipped with a default interface, that is the set of all the fragmentation points of the metamodel together with the corresponding target classes. The default interface exposes all the instances of the target class of some fragmentation point of the metamodel with respect to the corresponding association indicated by the fragmentation point.

More generally, module designer may specify the interfaces of their modules explicitly. However, to be meaningful, the specified interface should not expose more than the default interface allows. Formally, an interface specification of a module is defined as follows:

Definition 7 (Module interface)**.** *An interface of a module specifies a set of pairs of form $(a, i : C)$, where i is optional. Moreover, a is a fragmentation point of the metamodel in which the module is expressed and* $C \leq \mathsf{snd}(a)$.

From now on, we only consider metamodels with a non-empty set of fragmentation points, because according to the definitions above, models of metamodels with an empty set of fragmentation points expose nothing to the external for composition therefore are out of our interest.

6 Model Integration

Models that capture different subject matters of a system need to be composed together to render a system model functioning as a coherent whole. We realize model compositions by integrating the corresponding modules that encapsulate them via the interfaces. To do so, we first define fragments as the "glue" to join modules.

Definition 8 (Fragment)**.** *A fragment is a model that conforms to a fragment metamodel. Moreover, there is at least a referential instance (or place-holder).*

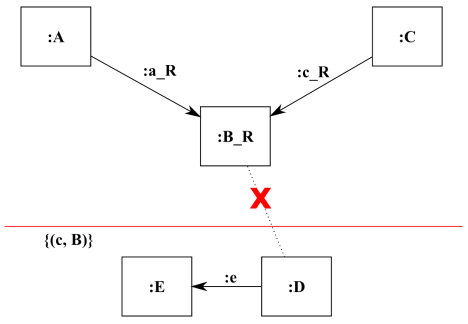
Following the construction of the fragment metamodel as given in Definition 5, we have the following property holds.

Property 4. *Given a fragment $F = (N_F, E_F, \mathcal{M}_F, \tau_F)$, for all referential links in it which lead to a referential target instance, i.e. $\forall e_F \in E_F$ such that $\tau_F(e_F) \in E^R_{\mathcal{M}_F}$ and $\tau_F(\mathsf{snd}(e_F)) \in N^R_{\mathcal{M}_F}$, we have $\mathsf{fst}(\mathsf{snd}(\mathsf{ori}(\tau_F(e_F)))) = \mathsf{ori}(\tau_F(\mathsf{snd}(e_F)))$.*

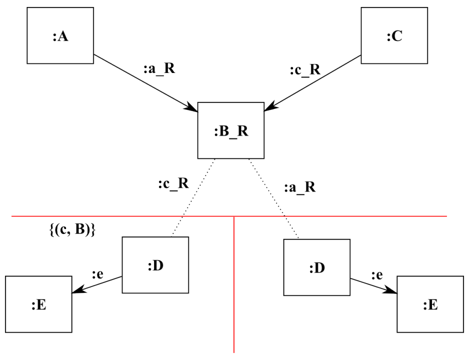
Informally, we integrate a set of modules via a fragment by mapping all the referential instances of the fragment to some instances of some participant modules. Moreover, the mapping should meet two conditions:

1. typing is respected, in the sense that the type of the mapped instance should be a sub-type of the type of the referential instance;
2. interfaces of the modules are respected, in the sense that if an instance is not exposed with respect to an association in the interface, it is forbidden to map any referential instance to it, where the referential instance is the target of a link of the referential counterpart of the association.

Continue with the example given in Section 4 suppose we have a fragment as specified above the line in the diagram below, and a module below the line whose interface exposes only one pair (B, c), while leaving the other possible pair (B, a) invisible. As a consequence, the attempt to map the instance of "B_R" to the instance of "D" in the module is forbidden although "D" is type compatible with "B_R" because "D" is a sub-class of its counterpart "B".



By contrast, the following integration attempt is allowed, where, the fragment on the top composes two modules: the referential instance is mapped to the "D" instance from the left module with respect to the "c_R" typed link, and to the "D" instance from the right module with respect to the "a_R" typed link, where the right module uses the default interface.



Learned from the example, the mapping of referential instances in the fragment needs to be done with respect to the referential link of which the instance is the target. More specifically, the same instance of "B_R" is mapped to two different "D" instances, with respect to the two links leading to it.

The semantics of integration is a new model, in which:

1. the set of nodes is the union of all the nodes of the participant modules, plus the non-referential ones of the fragment, because all the referential instances in the fragment are replaced by the mapped instances from some participant modules. The types of the nodes remain the same.

2. the set of edges is the union of all the edges of the participant modules, plus the variants of the edges of the fragment depending on the following cases:

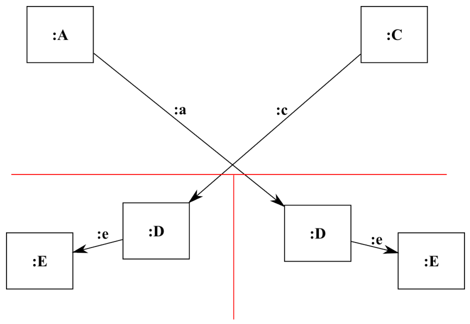
   (a) if the edge is an instance of an association of the original metamodel, then the edge remains with the same source, target, and type;

   (b) otherwise, i.e. the edge is an instance of an association that is introduced in the fragment metamodel, i.e. it is a referential counterpart of an original association, then we change the type of the edge to be the original association. Moreover,

      i. if the target of the edge is not a referential instance, then the edge remains with the same source and target;

      ii. otherwise, i.e. the target of the edge is a referential instance, we change the target into the mapped instance that is defined in some participant module while leaving the source of the edge the same.

The integrated model of the example above looks like the following:



Formally, we have the following definition of module integration.

Definition 9 (Module integration: syntax)**.** *An integration is defined over a set of valid modules* $\{M_1 : I_1, \ldots, M_k : I_k\}$*, via a valid fragment* $F = (N_F, E_F, \mathcal{M}_F, \tau_F)$*, with respect to a mapping function* ρ*, where:*

1. I_i *is the interface of module* M_i*,* $i = 1, \ldots, k$*.*
2. $M_i = (N_i, E_i, \mathcal{M}_i, \tau_i), i = 1, \ldots, k$*, have the same metamodel* $\mathcal{M}$*.*
3. *The metamodel of* F *i.e.* $\mathcal{M}_F$ *is the fragment metamodel* $\mathcal{M}$*.*
4. *Let* N_r *denote the set of referential instances in* F*, i.e. they are all instances of referential class nodes i.e.* $N^R_{\mathcal{M}_F}$*. Let* E_r *denote the set of referential links in* F*, i.e. they are all instances of referential association edges i.e.* $E^R_{\mathcal{M}_F}$*.*

   ρ *is a partial function:* $N_r \times E_r \to (N_1 \cup \ldots \cup N_k)$ *defined for all* (n_r, e_r) *pairs where* $n_r \in N_r$*,* $e_r \in E_r$ *and* $n_r = \mathsf{snd}(e_r)$*. Moreover,* ρ *satisfies the following two conditions. Let* $\rho(n_r, e_r) = n_i$*, where* $n_i \in N_{M_i}$ *for some module* M_i *of interface* I_i*. We have:*

   (a) *suppose* $\tau_F(n_r)$ *is the referential class node of a class node* $n_{\mathcal{M}}$ *in the metamodel* $\mathcal{M}$*, we have* $(\tau_i(n_i), n_{\mathcal{M}}) \in H^*_{\mathcal{M}}$*. Namely,* ρ *maps type compatible instances to referential instances;*

   (b) *suppose* $\tau_F(e_r)$ *is the referential association edge of an association* $e_{\mathcal{M}}$ *in the metamodel* $\mathcal{M}$*, we have* $(e_{\mathcal{M}}, n_i : n_{\mathcal{M}}) \in I_i$*. Namely,* ρ *maps visible instances to referential instances.*

Definition 10 (Model integration: semantics)**.** *The semantics of an integration returns a model* $M = (N, E, \mathcal{M}, \tau)$*, where:*

1. $N = (\bigcup_{i=1,\ldots,k} N_i) \cup (N_F \setminus N_r)$*, and*

$$\tau(n) = \begin{cases} \tau_i(n) & n \in M_i \\ \tau_F(n) & n \in N_F \end{cases}$$

2. $E = (\bigcup_{i=1,\ldots,k} E_i) \cup \{v(e_F) \mid e_F \in E_F\}$*, where* $\tau(e_i) = \tau_i(e_i)$ *for all* $e_i \in E_i$*, and*

(a) *if* $\tau_F(e_F) \in E^O_{\mathcal{M}_F}$, *then* $v(e_F) = e_F$ *and* $\tau(v(e_F)) = \tau_F(e_F)$;

(b) *otherwise, i.e.* $\tau_F(e_F) \in E^R_{\mathcal{M}_F}$,

   i. *if* $\mathsf{snd}(e_F) \notin N_r$, *then* $v(e_F) = e_F$;

   ii. *otherwise* $v(e_F) = (\mathsf{fst}(e_F), \rho(\mathsf{snd}(e_F), e_F))$.

   And in both cases $\tau(v(e_F)) = \mathsf{ori}(\tau_F(e_F))$.

Immediately following the definition above, we have the property below hold.

Property 5. *All nodes in the result model, i.e.* $\forall n \in N$, *are instances of some class of the original metamodel, i.e.* $\tau(n) \in N_{\mathcal{M}}$; *and all edges in the result model, i.e.* $\forall e \in E$, *are instances of some association of the original metamodel, i.e.* $\tau(e) \in E_{\mathcal{M}}$.

In other words, the result model is a well defined model according to Definition 2. Moreover, the following theorem shows that it is also a valid one, i.e. conforming to the specified metamodel.

Theorem 1. *The result model* $M = (N, E, \mathcal{M}, \tau)$ *of an integration over a set of disjoint* $M_i = (N_i, E_i, \mathcal{M}, \tau_i)$, $1 \le i \le k$, *via* $F = (N_F, E_F, \mathcal{M}_F, \tau_F)$, *with respect to* ρ, *is a valid model conforming to the metamodel* $\mathcal{M}$.

Proof. To prove that a model conforms to its metamodel, it amounts to checking against the two well-formedness rules listed in Definition 3.

Type compatibility Given an edge/link $e \in E$ in the result model, we discuss the cases following Definition 10.2.

1. The link comes from a participant module, that is $e \in E_i$, $1 \le i \le k$. The two ends of the link are type compatible, because the participant module M_i is a valid model of the same metamodel.

2. The link is a variant of a link of the fragment model, that is $e = v(e_F)$ for some $e_F \in E_F$.

   (a) $\tau_F(e_F) \in E^O_{\mathcal{M}_F}$. In this case, we have $e = e_F$ hence $\tau(\mathsf{fst}(e)) = \tau_F(\mathsf{fst}(e_F))$, and $\tau(e) = \tau_F(e_F)$. Because the fragment F is a valid model conforming to $\mathcal{M}_F$, we have $\tau_F(\mathsf{fst}(e_F)) \le \mathsf{fst}(\mathsf{fst}(\tau_F(e_F)))$, therefore, we get $\tau(\mathsf{fst}(e)) \le \mathsf{fst}(\mathsf{fst}(\tau(e)))$. Symmetrically, we have also $\tau(\mathsf{snd}(e)) \le \mathsf{fst}(\mathsf{snd}(\tau(e)))$.

   (b) $\tau_F(e_F) \in E^R_{\mathcal{M}_F}$. Following Definition 5, $\mathsf{snd}(\tau_F(e_F))$ must be a newly added abstract super class of a referential class and its original counterpart in $\mathcal{M}$.

      i. $\mathsf{snd}(e_F) \notin N_r$, therefore $\mathsf{snd}(e_F)$ must be an instance of a subclass of $\mathsf{snd}(\mathsf{ori}(\tau_F(e_F)))$, hence

$$\tau_F(\mathsf{snd}(e_F)) \le \mathsf{fst}(\mathsf{snd}(\mathsf{ori}(\tau_F(e_F)))) \tag{1}$$

Moreover, we also have $e = e_F$ and $\tau(e) = \mathsf{ori}(\tau_F(e_F))$ hence

$$\mathsf{fst}(\mathsf{snd}(\tau(e))) = \mathsf{fst}(\mathsf{snd}(\mathsf{ori}(\tau_F(e_F)))) \tag{2}$$

Meanwhile, following Property 2.1, we have

$$\mathsf{fst}(\mathsf{fst}(\tau(e))) = \mathsf{fst}(\mathsf{fst}(\mathsf{ori}(\tau_F(e_F)))) = \mathsf{fst}(\mathsf{fst}(\tau_F(e_F))) \tag{3}$$

Because F is valid w.r.t. $\mathcal{M}_F$, we have

$$\tau_F(\mathsf{fst}(e_F)) \leq \mathsf{fst}(\mathsf{fst}(\tau_F(e_F))) \tag{4}$$

As a consequence, we get

$$\begin{aligned} \tau(\mathsf{fst}(e)) = \tau_F(\mathsf{fst}(e_F)) &\overset{(4)(3)}{\leq} \mathsf{fst}(\mathsf{fst}(\tau(e))) \\ \tau(\mathsf{snd}(e)) = \tau_F(\mathsf{snd}(e_F)) &\overset{(1)(2)}{\leq} \mathsf{fst}(\mathsf{snd}(\tau(e))) \end{aligned}$$

ii. $\mathsf{snd}(e_F) \in N_r$. We have $e = (\mathsf{fst}(e_F), \rho(\mathsf{snd}(e_F), e_F))$, where $\rho(\mathsf{snd}(e_F), e_F)) \in N_j$, for some $1 \leq j \leq k$, and

$$\tau(e) = \mathsf{ori}(\tau_F(e_F)) \tag{5}$$

As a consequence, we have

$$\begin{aligned} \tau(\mathsf{fst}(e)) = \tau_F(\mathsf{fst}(e_F)) &\overset{(4)}{\leq} \mathsf{fst}(\mathsf{fst}(\tau_F(e_F))) \\ &\overset{(3)}{=} \mathsf{fst}(\mathsf{fst}(\mathsf{ori}(\tau_F(e_F)))) \overset{(5)}{=} \mathsf{fst}(\mathsf{fst}(\tau(e))) \\ \tau(\mathsf{snd}(e)) = \tau_j(\rho(\mathsf{snd}(e_F), e_F)) &\overset{Def.9.4a}{\leq} \mathsf{ori}(\tau_F(\mathsf{snd}(e_F))) \\ &\overset{Prop.4}{=} \mathsf{fst}(\mathsf{snd}(\mathsf{ori}(\tau_F(e_F)))) \end{aligned}$$

Multiplicity compatibility Given a node $n \in N$ in the result model, we discuss the number of links that are relevant for it, i.e. those either starting from or leading to n. Following Definition 10.2, n is of either the following cases.

1. The node comes from a participant module M_i, that is $n \in N_i$, $1 \leq i \leq k$. Because all the participant modules are disjoint, this node is not shared with other participant module M_j, i.e. $n \notin N_j$, $1 \leq j \leq k$ and $i \neq j$. We discuss the following two sub-cases:

   (a) The node is not hooked upon during integration, i.e. $n \notin \mathsf{range}(\rho)$. In this case, the links that are relevant for n in the result model is the same as they are in M_i. Because M_i is valid, no violation of multiplicity is introduced.

(b) The node is hooked upon during integration, i.e. $n \in \mathsf{range}(\rho)$. In this case, during the integration, extra links relevant for n in M are introduced. For any such link $e \in E \setminus E_i$, we have $\mathsf{snd}(e) = n$; $\exists! e_F \in E_F$, such that $\tau_F(e_F) \in E^R_{\mathcal{M}_F}$, and $\exists! n_F \in N_F$, such that $\mathsf{snd}(e_F) = n_F$ and $\tau_F(n_F) \in N^R_{\mathcal{M}_F}$; $\rho(n_F, e_F) = n$; and $\tau(e) = \mathsf{ori}(\tau_F(e_F))$. Following Property 3, we thus have $\mathsf{snd}(\mathsf{fst}(\tau(e))) = \mathsf{snd}(\mathsf{fst}(\mathsf{ori}(\tau_F(e_F)))) = (\_, \infty)$. As a consequence, the addition of e does not introduce violation of multiplicity.

2. The node comes from the fragment, that is $n \in N_F$. We discuss with respect to the following kinds of associations in $\mathcal{M}$.

   (a) $\forall e_{\mathcal{M}} \in E_{\mathcal{M}}$ where $e_{\mathcal{M}} \notin \mathsf{range}(\mathsf{ori})$, i.e. $e_{\mathcal{M}}$ does not have a referential counterpart in $\mathcal{M}_F$. In this case, all the relevant links to n of type $e_{\mathcal{M}}$ in the result model M are from the fragment F as such following Definition 10.2a. Because F is valid, the number of links remain multiplicity compatible.

   (b) $\forall e_{\mathcal{M}} \in E_{\mathcal{M}}$ where $e_{\mathcal{M}} \in \mathsf{range}(\mathsf{ori})$, i.e. $e_{\mathcal{M}}$ has a referential counterpart in $\mathcal{M}_F$. In this case, for all n relevant links e's in M of type $e_{\mathcal{M}}$, $\exists! e_F \in E_F$, such that $e_{\mathcal{M}} = \tau(e) = \mathsf{ori}(\tau_F(e_F)) = e^R_{\mathcal{M}}$. Moreover, following Property 2, the two associations $e_{\mathcal{M}}$ and $e^R_{\mathcal{M}}$ have the same multiplicities on both sides. Because F is valid, i.e. the set of e_F's is multiplicity compatible w.r.t. $e^R_{\mathcal{M}}$, so is the set of e's w.r.t. $e_{\mathcal{M}}$.

□

7 Model Hierarchies

We summarize the modeling activity of a system in a given metamodel following the modular modeling paradigm. We start from a set of disjoint unit modules, each of which is a valid model of the metamodel equipped with an interface and captures a separate subject matter of the system. A fragment is designed to integrate some of the unit modules with respect to a mapping function ρ, during which only the elements in the participant modules that are exposed in the interfaces are visible and manipulated. Following Theorem 1, the result of the integration is guaranteed to be a valid model conforming to the original metamodel, hence neither syntactical nor type checking of the result model, which is normally more complicated than the participant ones, is needed. In other words, modular modeling supports modular checking. Explicit interface may be specified for this result model or the default one is used, which can then recursively participate in further integrations until the system model is reached. As a consequence, we get the following structure of module hierarchies.

Definition 11 (Module Hierarchy)**.** *A module hierarchy is a directed acyclic graph whose nodes are either modules or fragments, such that:*

1. *all sink nodes (i.e., nodes with no outgoing edges) and source nodes (i.e., nodes without incoming edges) are modules;*
2. *each node that is a fragment has as successors all the nodes that represent the modules participating in the integration via the fragment;*
3. *each node that is a module has either no successors (representing a unit module), or it has as successor a single fragment (representing a module that is derived by integrating the fragment with its participants).*

Moreover, we have the following theorem hold.

Theorem 2. *All the models that correspond to the module nodes in a module hierarchy are valid models conforming to the original metamodel.*

Proof. We catogerize the models represented by the module nodes in a module hierarchy into three kinds:

1. unit models: These are the models corresponding to the unit modules of the hierarchy, namely, those of the sink nodes;
2. level-one models: These are the models corresponding to the module nodes that have as successor a single fragment node whose successors are all unit modules;
3. level-n models: These are the models corresponding to the module nodes that have as successor a single fragment node among whose successors there is at least one that is not a unit module.

We prove the theorem according to the three cases listed above:

unit models: Trivial.

level-one models: True following Theorem 1 because all the unit models are disjoint.

level-n models: To prove that a model conforms to its metamodel, it amounts to checking against the two well-formedness rules listed in Definition 3, i.e. type compatibility and multiplicity compatibility.

Type compatibility The same set of arguements in the proof of Theorem 1 for this purpose can be re-used.

Multiplicity compatibility Because the participant models of a level-n model may share nodes, two extra cases need to be discussed. More specifically, we follow the line of proof of Theorem 1 for this purpose, where we examined the multiplicities influenced by the relevant links of a given node n in the result model (i.e. the level-n model under consideration here), accroding to two cases, where in the first case, two sub-cases need to be discussed in addition, as follows.

1. The node comes from a participant module M_i;

(a) The node is not shared with other participant module M_j, i.e. $n \notin N_j$, $1 \leq j \leq k$ and $i \neq j$, neither is the node hooked upon during integration, i.e. $n \notin \mathsf{range}(\rho)$. Proof as in Theorem 1.

(b) The node is not shared with other participant module M_j, i.e. $n \notin N_j$, $1 \leq j \leq k$ and $i \neq j$, but it is hooked upon during integration, i.e. $n \in \mathsf{range}(\rho)$. Proof as in Theorem 1.

(c) The node is shared with another participant module M_j, i.e. $n \in N_j$, $1 \leq j \leq k$ and $i \neq j$, but it is not hooked upon during integration, i.e. $n \notin \mathsf{range}(\rho)$. In this case, the relevant links of n in both M_i and M_j are included into the result model.
For those with identical opposite ends, because only one copy remains in the result model, hence the number of links remains the same, hence no multiplicity violation is introduced.
For two n relevant links that have distinguished opposite ends n_i and n_j in M_i and M_j respectively, both the two links remain in M. Only in cases where the two links both have n on the source (or target) end, and have the same type in the result model M, there is a potential possibility to violate the multiplicity of that association.
In modular modeling paradigm, we always start from disjoint unit modules that are developed by separate teams in parallel. As a consequence, because M_i and M_j are not disjoint, at least one of them, say M_j, is not unit, but is rather the result model of a sequence of integrations. Moreover, following Definition 9 and 10, among these integration steps, there must exist and only exist one integration during which n is in a participant module, n_j is in the fragment, and the link between n_j and n is introduced by replacing some referential link.
To summarize, the analysis above tells us the following two points:

i. it is possible for n to be the same target of the two links, but not the same source;

ii. according to Property 3, we know that the multiplicity of the source end has an infinite upper bound.

As a consequence, there is no multiplicity violation introduced in this case either.

(d) Finally, the node is shared with another participant module M_j, i.e. $n \in N_j$, $1 \leq j \leq k$ and $i \neq j$, so is it hooked upon during integration, i.e. $n \in \mathsf{range}(\rho)$. In this case, the influence brought by the integration on the numbers of link ends for an association can be considered as the accumulation of the impacts brought by the two previous cases above. As neither of them introduce violation, nor is this case.

2. The node comes from the fragment. Proof as in Theorem 1.

□

The system model is designated by the union of all the top models, i.e. those corresponding to the source nodes in the hierarchy. Moreover, the system model is also a valid model conforming to the original metamodel.

Theorem 3. *The union of all the models that correspond to the source nodes in a module hierarchy is a valid model conforming to the original metamodel.*

Proof. This theorem holds as a corollary of Theorem 2, if we consider there exist a special empty fragment that integrates all these top models. □

We introduce a more compact representation of module hierarchies called integration hierarchies.

Definition 12 (Integration hierarchy)**.** *The integration graph for a module hierarchy is obtained by collapsing each edge in the module hierarchy graph that leads from a module node to a fragment node into the fragment node.*

Property 6. *For a given integration hierarchy, joining any two nodes, (which implies joining the corresponding fragments and/or modules of the nodes), will still render an integration hierarchy, where the result node is a fragment node if there is still outgoing edges from it, otherwise, a module node.*

The union-based composition semantics is straightforward to understand. It is non-destructive because all the unit modules appear directly as such in the system model. Moreover, even if there are modules that are re-used in multiple places during the sequences of integrations, thanks to the union-based semantics of integration, duplication is avoided.

Note that the standing point of this work is not to provide yet another new approach to take over the whole model composition area, but rather to provide a complementary approach to existing ones, with which we believe that a large number of interesting problems can be tackled in a more straightforward, elegant, and light-weighted manner than applying other existing approaches. We admit the existence of grey zones in which our approach lacks enough power. However, our approach is as expressive and powerful as the host metamodel offers, which can be substantial.

8 Model Decomposition

Definition 13 (Fragmentable link)**.** *Given a model $M = (N, E, \mathcal{M}, \tau)$, a link $e \in E$ is fragmentable if there exists a fragmentation point of $\mathcal{M}$ written a such that $\tau(e) = a$.*

We denote the set of fragmentable links by $E^f \subseteq E$. Intuitively, a fragmentable link denotes a point in the model where the target end of the link can appear in a separate module.

Consequently, we have the following definition of non-fragmentable paths in a model. Intuitively, a non-fragmentable path consists of none fragmentable links.

Definition 14 (Non-fragmentable path)**.** *Given a model $M = (N, E, \mathcal{M}, \tau)$, a non-fragmentable path is a sequence of links $e_1; \ldots ; e_k$, such that:*

1. $e_i \notin E^f$, $1 \leq i \leq k$;
2. $\mathsf{snd}(e_i) = \mathsf{fst}(e_{i+1})$, $1 \leq i \leq k$;

8.1 Model slicing

Given a model $M = (N, E, \mathcal{M}, \tau)$, model slicing returns a slice of M, called S. From the graph point of view, S is a sub-graph of M, i.e. a slice S takes the form: (N_s, E_S), where $N_s \subseteq N$ and $E_S \subseteq E$.

Algorithm: Slice(M)

1. Pick a node from M, i.e. $n \in N$. Put n into S, i.e. $n \in N_S$.
2. $\forall n' \in N, n' \neq n$, such that there exists a non-fragmentable path between n and n' (either from n to n' or from n' to n), put n' into S, together with the links in the non-fragmentable path.
3. Repeat step 2 for all newly added nodes in S, until there is no new node introduced into S.
4. $\forall e \in E^f$, if $\mathsf{fst}(e) \in N_S$, then also add $\mathsf{snd}(e)$ and e into S.

8.2 Model fragmentation

Given a model M, the fragmentation of it keeps slicing sub-graphs from the model until there is no further slicing possible. As a consequence, model fragmentation returns a set of slices, $S_1, \ldots, S_k$. From the graph point of view, S_i's are all sub-graphs of M.

Algorithm: Fragment(M)

```
1. $i = 1, M_{in} = M, S_1 = \text{Slice}(M)$.
2. While $S_i \neq M_{in}$ do
   (a) $M'_{in} = (N'_{in}, E'_{in}, \mathcal{M}, \tau)$, where
       • $E'_{in} = E_{in} \setminus E_{S_i}$,
       • $N'_{in} = \{n \mid n \in N_{in}, \exists e \in E'_{in}$ s.t. $n \in \{\mathsf{fst}(e), \mathsf{snd}(e)\}\}$;
   (b) $i' = i + 1$;
   (c) $S_{i'} = \text{Slice}(M'_{in})$
```

8.3 Model decomposition

The set of slices we get at the end of the fragmentation of a model is not very informative as such, because we do not know yet if they conform to any metamodel and to which one. Ideally, the result of modularizing a model is to reach a set of sub-models conforming to the same original metamodel, and a set of fragments conforming to the fragment metamodel of the original metamodel such that all together constitutes an integration hierarchy as defined in Definition 12. As a consequence, these two sets give us a meaningful way of breaking down the original model in the sense that the sub-models can be considered as unit modules each of which captures a specific subject matter, and the fragments offer the "glue" between these modules that allows to incrementally build the overall model.

Towards this direction, we first examine the slices got as the result of the fragmentation of a model M with respect to the original metamodel $\mathcal{M}$.

Because a slice S is a sub-graph of the original model M, the fragmentable links are inherited from M, i.e. a link of S is fragmentable if it is so in M. Similarly, the notion of non-fragmentable paths in slices can also be derived, i.e. a non-fragmentable path in S consists of only non-fragmentable links.

Intuitively, if two nodes are connected by both a fragmentable link and a (sequence of) non-fragmentable path(s), where the former indicates the target node can be separated into another module (which is a permission) while the latter forbids so (which is a prohibition), in order to satify both, the target node should be prohibited from being separated into a different module. The following definition excludes such "false" fragmentable links in a slice by identifying those real ones.

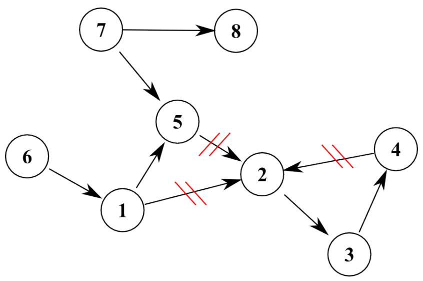
Definition 15 (Real fragmentable link)**.** *Given a slice $S = (N_S, E_S)$, we say a link $e \in E_S$ is real fragmentable if it is fragmentable, and moreover the binary relation R between nodes does not hold for* $\mathsf{fst}(e)$ *and* $\mathsf{snd}(e)$*, where the binary relation $R \subseteq N_S \times N_S$ is defined as follows:*

1. $r = \{(n_1, n_2) \mid$ *exist a non-fragmentable path* $e_1; \ldots; e_k$ *in* S *s.t* $\mathsf{fst}(e_1) = \mathsf{fst}(e)$ *and* $q\mathsf{snd}(e_k) = \mathsf{snd}(e)\}$*;*
2. $R = (r \cup r^{-1})^+$

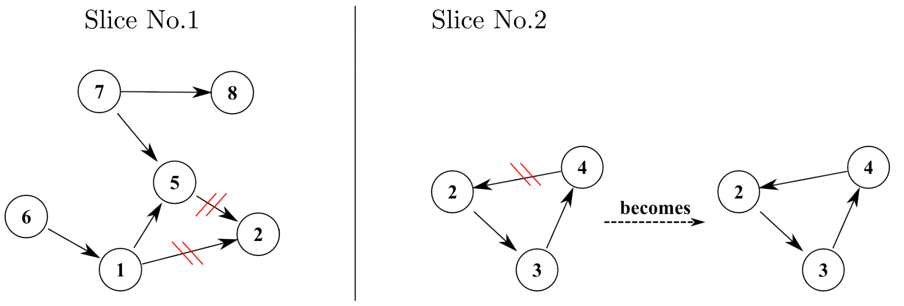
Property 7. *For all the slices got at the end of fragmenting a model M:*

1. *the slice is a model of the original metamodel if there is no real fragmentable links in it, and we call it a module;*
2. *otherwise, the slice conforms to the fragment metamodel of the original metamodel, (where the targets of all the real fragmentable links are replaced by corresponding "place holders", i.e. instances of referential classes), and we call it a fragment.*

For illustration purpose, consider the following example model, where all fragmentable links are indicated by two parallel short red cuts.



After the fragmentation, we get two slices as follows:



Examing the two slices against the Definition 15 above, we conclude that the fragmentable link in the second slice from node 4 to 2 is a false one because there exists a non-fragmentable path between the two nodes via node 3. As a consequence, the first slice is a fragment while the second slice is a module. From now on, only real fragmentable links remain fragmentable.

Treating all the derived fragments and modules themselves as nodes, we then draw a slice graph of them, accroding to the following definition.

Definition 16 (Slice graph)**.** *A slice graph is a directed graph whose nodes are either fragments or modules. A fragment node (F_1) has a outgoing edge to another node, fragment (F_2) or module (M), if and only if there is a node in F_1, which is the target of a real fragmentable link in F_1, and appears also in F_2 (or M) while without being the target of a real fragmentable link in the latter.*

Note that modules do not have outgoing edges because there is no real fragmentable links in them.

In comparison to the graphs corresponding to integration hierarchies, slice graphs differ in the sense that there might be circles of fragment nodes in them. The fragments on a circle are mutually dependent in the following sense:

Definition 17 (Mutual fragments)**.** *A set of slices $S_1, \ldots, S_k$ is a set of mutually complementary fragments if and only if the set itself but none of its sub-sets satisfies the following two conditions:*

1. *S_i, $1 \leq i \leq k$, are all fragments.*

2. *For all real fragmentable links $e \in E_{S_i}$, $1 \leq i \leq k$, there exists a slice S_j, $i \neq j$ and $1 \leq j \leq k$, such that:*

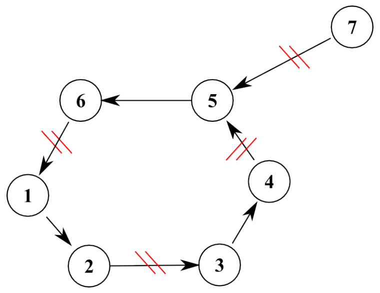
   (a) $\mathsf{snd}(e) \in N_{S_j}$*;*

   (b) $\nexists e' \in E_{S_j}$ *such that* $\mathsf{snd}(e) = \mathsf{snd}(e')$ *and e' is real fragmentable.*

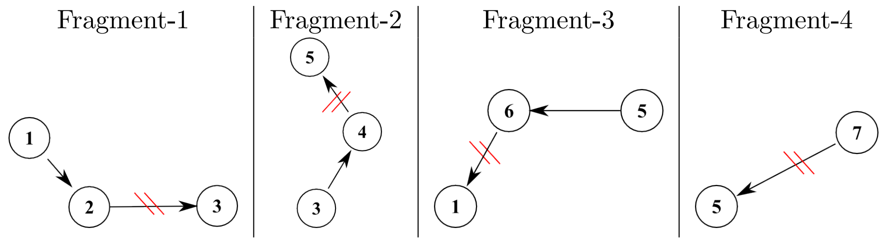
In order to reach the integration hierarchy of a model, such mutually dependent fragments on a circle need to be combined into one piece. Moreover, we have the following property hold:

Property 8. *The union of the set of mutual fragments is a module conforming to the original metamodel.*

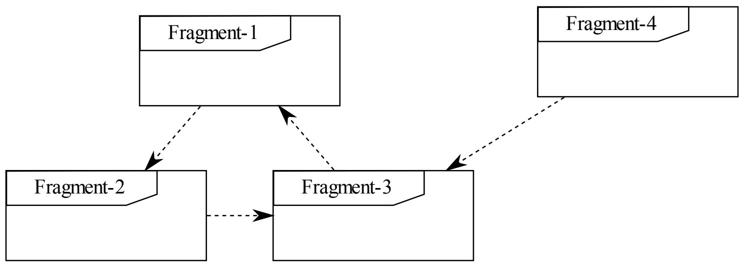
More concretely, consider the following example model.



After the fragmentation, we get four slices as follows:



All the four slices are fragments. The corresponding slice graph looks like:



where the three fragments: Fragment-1, Fragment-2 and Fragment-3 constitutes a circle hence the three fragments are mutually dependent and have to be combined into one module.

Note that the reason why we enforce the sets of mutual fragments in Definition 17 to be the smallest is because we want to achieve the degree of decomposition as much as possible. In the example above, both the two sets: Fragment-1, Fragment-2 and Fragment-3; and Fragment-1, Fragment-2, Fragment-3 and Fragment-4, satisfy the two conditions listed in Definition 17 hence could potentially be considered as mutual fragment sets. However, if we go for the second set, the result of decomposition will be a unique module that is the original model itself, which is not as interesting as if we go for the first set, where we achieve a finer degree of decomposition that consists of a module (the union of Fragment-1, Fragment-2 and Fragment-3) and a fragment (Fragment-4).

In summary, to modulariza a model, we first get the set of slices by applying the algorithm **Fragment()** to the model. Then the set of slices are examined to identify fragments and modules. As a final step, all mutual fragments are combined together into a module. The result of the decomposition of a model is a set of fragments and a set of modules, which together constitute an integration hierarchy by connecting the same node that appears in different pieces.

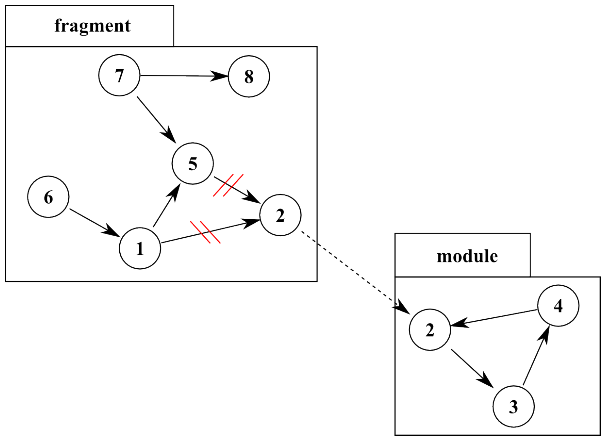
Algorithm Decomposition(M)

```
1. Fragment($M$) = $\{S_1, \ldots, S_k\}$.
2. haracterize $S_i$, $1 \leq i \leq k$, as a fragment or a module by identifying the
   real fragmentable links in it following Definition 15. As a consequence,
   the set of slices $\{S_1, \ldots, S_k\}$ is divided into two disjoint sub-sets, one for
   modules denoted by $MS$, the other for fragments denoted by $FS$.
3. Detect the sets of mutual fragments in $FS$ following Definition 17, and
   combine them into modules.
   (a) $\Pi = \emptyset$, $\Sigma = FS$, $i = 2$;
   (b) While $\sharp\Sigma \leq i$ do
       i. let $\Sigma_i$ be the set of all sub-sets of $\Sigma$ with size $i$, i.e. $\Sigma_i \subseteq \mathcal{P}(\Sigma)$
          and $\forall\sigma \in \Sigma_i$, $\sharp\sigma = i$;
       ii. While $\Sigma_i \neq \emptyset$ do
           A. let $\sigma$ be a memeber of $\Sigma_i$;
           B. if $\sigma$ is a set of mutual fragments following Definition 17;
              then
              • $\Sigma' = \Sigma \setminus \sigma$;
              • $\Pi' = \Pi \cup (\bigcup_{F \in \sigma} \sigma)$;
              • $\Sigma'_i = \{\sigma' | \sigma' \in \Sigma_i, \sigma \cap \sigma' = \emptyset\}$
              else $\Sigma'_i = \Sigma_i \setminus \{\sigma\}$
       iii. $i' = i + 1$;
```

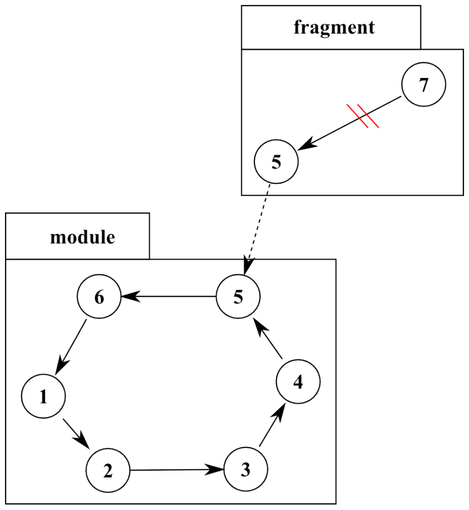
The final result of this step is a new set of modules: $MS' = MS \cup \Pi$; and a new set of fragments: $FS' = \Sigma$, which constitute the pieces for building the integration hierarchy.

4. If a node in a piece is the target of a real fragmentable link, and the same node also appears also in another piece while without being the target of a real fragmentable link in the latter, connect the first node to the second node.

As an example, after applying **Decomposition()** to the model appearing on page 22, the derived integration hierarchy looks like the following:



And the integration hierachy after applying **Decomposition()** to the model appearing on page 23 looks like the following:



9 Related Work

Model composition is a technique for building bigger models from smaller models, thus allowing system designers to control the complexity of a model-driven design process. Although model composition is a fundamental technique in model-driven engineering, it has received less attention from the research community than the related technique of model transformations. It is a commonly accepted view that there is a close link between model composition and model transformations in the sense that model composition can be viewed as or implemented by a model transformation.

Only fairly recently a common set of definitions for model composition was proposed and a set of requirements for model composition languages and tools was derived [3]. The definitions in that paper are based on examining three model composition frameworks: the Glue Generator Tool [3, 4], the Epsilon Merging Language [10], and the Atlas Model Weaver [1].

Much of the work on model composition has been developed within the area of aspect-oriented modelling (AOM). Some approaches to model composition in the AOM community focus only on integrating structural aspects in the composed model. For example France et al. [13] have developed a systematic approach for composing class diagrams in which a default composition procedure based on name matching can be customized by user-defined composition directives. Other authors have attempted to compose both structural and behavioural aspects. For instance in the work proposed by [6] we are given (1) a primary model, (2) aspect models and the bindings used to instantiate them in the application context, and (3) composition directives that determine how the instantiated aspect models are composed with the primary model to produce a composed model. A primary model consists of UML classifier and interaction diagrams, reflecting both structural and behavioural aspects.

The approach in [6] is an example of an asymmetric composition approach [2]: it is based on weaving an aspect concern with a base concern. Another approach to model composition is based on symmetric compositions, in which the composed models are at the same level. Examples are the Hyperspace approach [14], Subject Oriented Design [5], and the RAM approach [9]. Symmetric model composition techniques are typically based on merging the component models; this involves identifying common elements in the component models and defining how they should be fused [2].

The model composition techniques described above may be called white-box model composition techniques: they are usually based on having full access to the modelling elements within each model. A model composition technique with a more modular flavor that treats the component models as black boxes was defined in [11]: their approach, named the Collaborative Component Based Model approach (CCBM) leverages software component principles and focuses on the specification of how models collaborate with each other. The Collaborative Component Based Model approach achieves black-box reuse of unmodified models and preserves them. Thus, in CCBM, models are units of reuse and integration is modular and incremental, just as for software components in Component Based Software Engineering (CBSE) [7].

In our paper we propose a model composition technique similar in spirit to the CCBM approach described above. It is also not based on transforming the component models but rather provides additional plumbing - the fragments -

that connects the component models without changing them. Our approach differs from the CCBM approach in two ways: first the glue models used for composing models have a metamodel (the fragment metamodel) closely related to the metamodel of the participant models while in the CCBM approach another language (JPDD) is used for specifying the glue between the participant models. Furthermore their composition mechanism is less general in the sense that it only addresses how operations of the participant models collaborate.

Another work [15] addresses the problem of information hiding at the level of metamodels that are instances of MOF. This contrasts with our approach that concerns itself with information hiding at the level of models rather than metamodels. That is, the focus of our paper is to compose models into larger models rather than composing languages.

The authors of [8] develop a theory of model interfaces and interface composition in the context of dealing with soft references across XML models. Their definition of model interfaces is heavily influenced by the assumption that models are stored as XML files: their interfaces are based on the attribute names of the XML models and are not applicable to the more general setting subsumed by our approach.

10 Conclusion

In this paper we have presented a modular technique for composing models conforming to the same metamodel. The modularity of the approach comes from equipping the models participating in the composition with an interface, essentially a subset of the modeling elements in each model. The model composition only refers to the interfaces of the participant models. This provides reduced coupling between the participant models and the composed model, which facilitates the comprehension and maintenance of the composed model.

Another line of application of our modular model composition techinique concerns the model comprehension aspects. The benefits for model comprehension are addressed by the reverse process of building model hierarchies. That is given an existing model, we decompose it into a model hierarchy, thereby facilitate the comprehension of the initial model.

In this paper we have only taken into account metamodels that are expressed visually using the class diagram notation. Further textual constraints (expressed for instance in OCL) have not been taking into account. Considering additional constraints not expressed visually will likely lead to a more complicated definition of fragmentation points. This will be the subject of future work.

References

[1] Atlas model weaver project, 2005. `http://www.eclipse.org/gmt/amw/`.

[2] Olivier Barais, Jacques Klein, Benoit Baudry, Andrew Jackson, and Siobhan Clarke. Composing multi-view aspect models. In *the Proceedings of the*

Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008), pages 43–52. IEEE Computer Society, 2008.

[3] Jean Bézivin, Salim Bouzitouna, Marcos Del Fabro, Marie P. Gervais, Frédéric Jouault, Dimitrios Kolovos, Ivan Kurtev, and Richard F. Paige. A canonical scheme for model composition. In *the Proceedings of 2nd European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA 2006)*, volume 4066 of *Lecture Notes in Computer Science*, pages 346–360, 2006.

[4] Salim Bouzitouna and Marie-Pierre Gervais. Composition rules for PIM reuse. In *the Proceedings of 2nd European Workshop on Model Driven Architecture with Emphasis on Methodologies and Transformations (EWMDA04)*, pages 36–43, 2004.

[5] Siobhán Clarke. Extending standard uml with model composition semantics. *Science of Computer Programming*, 44(1):71–100, 2002.

[6] Robert B. France, Indrakshi Ray, Geri Georg, and Sudipto Ghosh. Aspect-oriented approach to early design modelling. *IEE Proceedings - Software*, 151(4):173–186, 2004.

[7] George T. Heineman and William T. Councill. *Component-Based Software Engineering: Putting the Pieces Together (ACM Press)*. Addison-Wesley Professional, 2001.

[8] Anders Hessellund and Andrzej Wasowski. Interfaces and metainterfaces for models and metamodels. In *MoDELS '08: Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*, pages 401–415, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] Jörg Kienzle, Wisam Al Abed, and Klein Jacques. Aspect-oriented multi-view modeling. In *the Proceedings of 8th International Conference on Aspect-Oriented Software Development, (AOSD 2009)*, pages 87–98, 2009.

[10] D.S. Kolovos. Epsilon project. `http://www.cs.york.ac.uk/~dkolovos`.

[11] Audrey Occello, Anne-Marie Dery-Pinna, Michel Riveill, and Günter Kniesel. Managing model evolution using the CCBM approach. In *the Proceedings of 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS-MBD workshop)*, pages 453–462. IEEE Computer Society, 2008.

[12] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, December 1972.

[13] Y. R. Reddy, Sudipto Ghosh, Robert B. France, Greg Straw, James M. Bieman, N. McEachen, Eunjee Song, and Geri Georg. Directives for composing aspect-oriented design class models. *Transactions on Aspect-Oriented Software Development I*, pages 75–105, 2006.

[14] Peri L. Tarr, Harold Ossher, William H. Harrison, and Stanley M. Sutton Jr. *N* degrees of separation: Multi-dimensional separation of concerns. In *the Proceedings of 1999 International Conference on Software Engineering (ICSE'99)*, pages 107–119, 1999.

[15] Ingo Weisemöller and Andy Schürr. Formal definition of mof 2.0 metamodel components and composition. In *MoDELS '08: Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*, pages 386–400, Berlin, Heidelberg, 2008. Springer-Verlag.