A Modular Model Composition Technique

Pierre Kelsen and Qin Ma

Laboratory for Advanced Software Systems
University of Luxembourg
6, rue Richard Coudenhove-Kalergi
L-1359 Luxembourg
{Pierre.Kelsen, Qin.Ma}@uni.lu

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. However many current model composition techniques are themselves complex in the sense that they merge the internal elements of the participating models in non-trivial ways. In this paper we apply some of the ideas from modular programming to reduce the complexity of model compositions. Indeed we propose a model composition technique with a modular flavor that treats the participating models as black boxes. Our technique has several desirable features: it is simple, it does not require a separate language for expressing the composition, and the understanding of the resulting composed model is made easier by the modular nature of the model composition.

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 the large models should become easier to understand and to maintain. Most current model composition techniques permit the specification of rather complex composition operations. This is shown in the fact that many techniques are based on a separate language for specifying the composition (such as weaving models in AMW [1] or the Epsilon Merging Language in [10]). Because these model compositions can be complex, separate model transformations need to be defined to perform them: in [1], for instance, this transformation is generated from the weaving model. Furthermore a system specified by using complex model compositions is difficult to understand.

The main contribution of this paper is a model composition technique that has the following desirable features:

- it is simple: the composition is specified by mapping elements of one distinguished model (called a fragment) to elements of the other models;

– it is modular: the participating models in the composition are equipped with an interface: only the elements in the interface can be mapped to by the fragment
– no additional language is required to express compositions
– it is formally defined

The presentation of this paper is as follows: in the next section we present a running example that will serve to illustrate the concepts in this paper. 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 that will support modularity via information hiding. We present our model composition technique in section 6. We show that this technique yields hierarchies of models in section 7. The final two sections discuss the contributions and put them into the context of existing work (section 8) and present concluding remarks (section 9).

2 A Running Example: the EP Language

In this paper we illustrate our model composition technique using the EP modeling language. EP is a language that allows the specification of the structure and behavior of a software systems at a platform-independent level [6, 8, 7]. The central concepts are *events* - modeling elements that are used to model behavior - and *properties* - which are use to model the structure of the state. A metamodel of the EP language is given in figure 1. We will also use a model conforming to this metamodel for illustrative purposes: the model describes a document management system. This model will be first introduced in section 6.

3 Basic Definitions: Models and Metamodels

To formally define our model composition technique, we first need to give formal definitions of metamodels, models and model conformance. We extend the definitions of [2] by formalizing both models and metamodels as graphs, and the mapping between a model to its metamodel as graph morphism.
Conventions:

1. In the following formal narrations, for any pair p, we use $\mathsf{fst}(p)$ to denote its first element and $\mathsf{snd}(p)$ to denote its second element.
2. For any function f, we use $\mathsf{range}(f)$ to denote its co-domain.

3.1 Metamodels

A metamodel consists of a finite set of classes and a finite set of associations that are basically relations between classes - either associations or inheritance relations.

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) $\mathsf{snd}(h)$.*

An example of a metamodel is that of the EP-language given in figure 1 (ignore for the moment the fragmentation edges indicated by the parallel lines intersecting the associations).

3.2 Models

Models are built by instantiating the constructs, i.e. classes and associations, of a metamodel.

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: $(N \rightarrow N_{\mathcal{M}}) \cup (E \rightarrow E_{\mathcal{M}})$. 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 "conform to" the metamodel: typing and multiplicity constraints need to be respected.

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 with (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.

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

As an example of model conformance consider the model named *Log* in the upper right part of figure 4: it conforms to the metamodel of the EP-language given in figure 1 (if we ignore the interface definition).

4 Fragment Metamodels

Our model composition approach will make use of partial models as the glue to unite the participant models. These partial models, which will be called *fragments*, have external links to other models. At the level of the metamodel we need to indicate which associations can be instantiated into external links of fragments. For this purpose we introduce the notion of fragmentation edges.

Definition 4 (Fragmentation edges of a metamodel). *A fragmentation edge 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.*

In other words an association edge in the metamodel is a fragmentation edge if the maximum multiplicity of its source is not constrained. The intuition behind this definition is as follows: if an association a from node A to node B is a fragmentation edge then any B-instance can be the target of an arbitrary number of links (instances of this association) from A-instances. Thus if we add to existing models external links (instances of association a) from a fragment this will not violate the multiplicity constraint of the association. In a later section (section 6) this observation will be instrumental in proving that the result of the composition with a fragment is a model conforming to the original metamodel.

Figure 1 gives an example of a metamodel for the executable modeling language EP (from [7, 8]) together with nine fragmentation edges indicated by parallel lines crossing the associations. These fragmentation edges have been identified in accordance with the above definition. For instance the association named *target* from *ImpactEdge* (right side of diagram) to *LocalProperty* (at the top of the diagram) has an unconstrained source multiplicity (indicated by '*') - it is therefore marked as a fragmentation edge.

The central concept of our model composition technique is that a *fragment* which is essentially a partial model that has some external links that are typed by fragmentation edges. Because of these external links fragments of a model conforming to a metamodel MM do not conform to MM. In order to be able to treat fragments also as models (which is desirable in a model-driven approach) we introduce the notion of a *fragment metamodel*. The basic idea is to replace the external links of a fragment by links to *referential nodes* which represent a node in another model. At the level of metamodels this is achieved by replacing the endpoint of each fragmentation edge by a new class having two subclasses: one subclass represents normal instances of the original endpoint class while the other subclass represents referential nodes. A simple example of a fragment

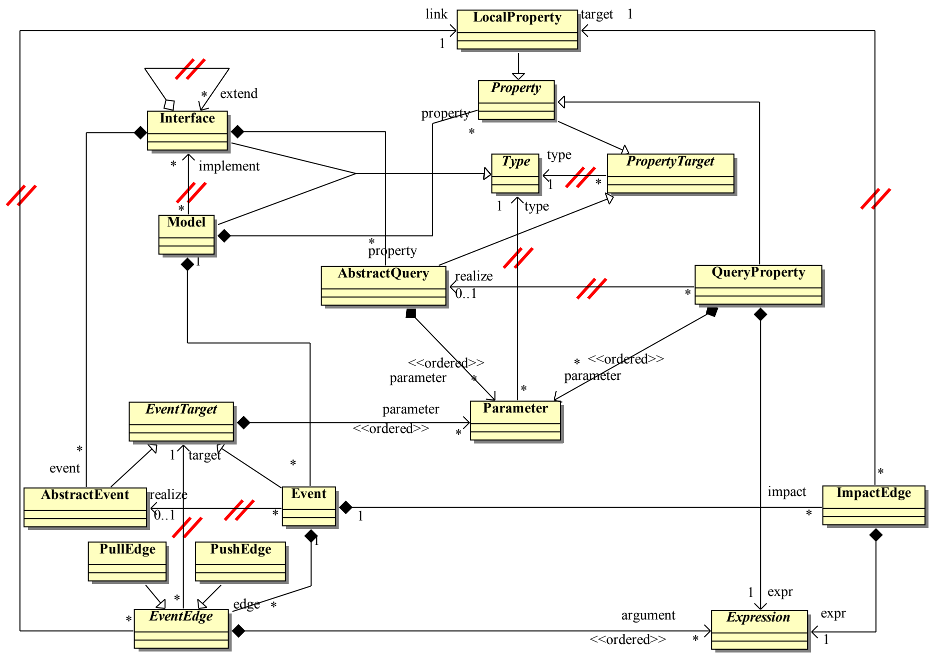

Fig. 1. A metamodel and its fragmentation edges

metamodel is given in figure 2. On the left side of that figure a metamodel is shown with two fragmentation edges. On the right side the corresponding fragment metamodel is given. A more involved example is shown in figure 3. This example will be discussed in more detail at the end of this section.

After this informal discussion we define fragment metamodels formally.

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 edge 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.*

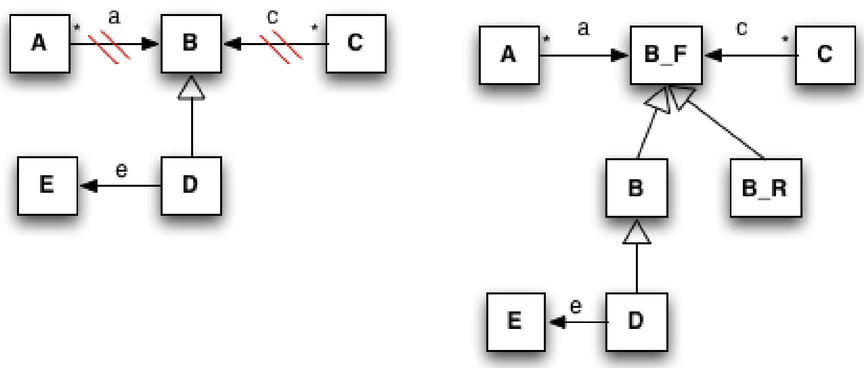

Fig. 2. A metamodel and its fragment metamodel

(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 from figure 1 is given in figure 3. We highlight all the changes with respect to the original metamodel in figure 1: The target class of a fragmentation edge in the original metamodel is now extended into a set of three classes indicated by the dashed contours, in which two new classes, namely, the referential counterpart class following the naming convention "XXXR" where "XXX" is the original name, and the common abstract super class with name "XXXF", are added. Moreover, the targets of fragmentation edges in the original metamodel are modified accordingly, leading to the newly added abstract super classes. These modified associations are highlighted as well in the diagram.

Definition 6 (Fragment). *A fragment is a model that conforms to a fragment metamodel. Moreover, there is at least one referential instance (or place-holder).*

5 Model Interfaces

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 disjoint pieces called modules that expose only a small subset of their internal elements to the other modules via interfaces. Modular programming offers several advantages: modules can be developed and tested independently, they can be reused more easily and they can be changed in more flexible ways.

In this section we apply the principle of information hiding to models. We do this by equipping the models with an interface which will be defined below.

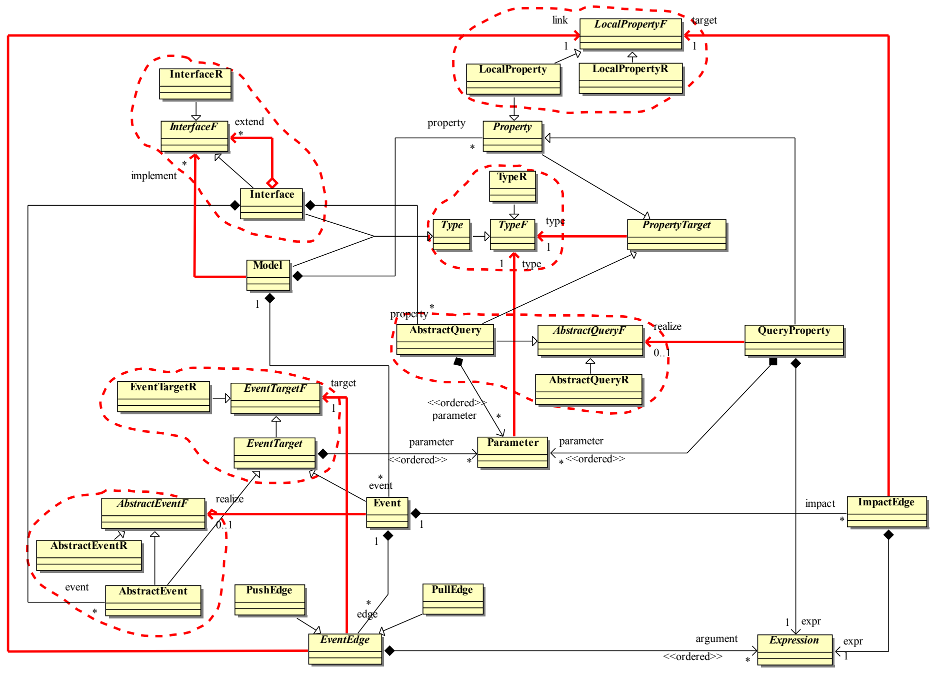

Fig. 3. A fragment metamodel of the EP metamodel

Definition 7 (Modules). *A module is a model with an interface.*

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, all instances in the module that are of type C are considered.

Definition 8 (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 edge of the metamodel in which the module is expressed and $C \leq \mathsf{snd}(a)$.*

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

If we view models as components, it is worthwhile noting that our approach differs from the classical definition of information hiding in component-based systems in terms of import and export interfaces. In our framework models simply do not have import interfaces but only export interfaces (as defined above). That is, models cannot access features that they do not "implement"; only fragments have this capability.

On the right side of figure 4, we show two modules - named Document and Log - of the EP metamodel (from figure 1) that are part of a document management system: their interfaces are shown at the bottom of each model. As an example consider the pair $(type, Document : Interface)$ that is part of the interface of the *Document* module. The presence of this pair means that a fragment can have as external link a type link to the *Document* interface of the *Document* module. Here *type* is the fragmentation edge from *PropertyTarget* (a superclass of *Property*, itself a superclass of *LocalProperty*) to *Type* in the EP-metamodel (see at the top of figure 1). In other words a fragment can use elements of type *Document*; in the example we see indeed that the fragment *DocumentLog* uses a property *document* of type *Document*.

6 Model Integration

We call our model composition technique *model integration*. We perform model integration using a set of modules and a fragment by mapping all the referential edges of the fragment to instances of some participant modules. Moreover, the mapping should meet two conditions (which will be more precisely defined in definition 9):

1. typing is respected, in the sense that the type of the target of the mapped referential edge 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 a referential edge to it that is typed by the referential counterpart of the association.

An example of a model integration scenario is shown in figure 4. In this example the fragment (shown at left) has five external links into the Log module and one external link into the Document module. The referential instances of the fragment, which are the source nodes of the dashed arrows, are all mapped to some type compatible instances that are exposed in the interfaces of the participant models.

Module integration is formally defined as follows.

Definition 9 (Integration Mapping). *An integration mapping of a fragment F over a set of modules $\{M_1 : I_1, \ldots, M_k : I_k\}$ having a common metamodel $\mathcal{M}$ is a function mapping each referential edge $e_r = (n, n_r)$ of the fragment to some instance n_i of some model M_i such that*

1. *if the type of n_r is the referential class node of a class node $n_{\mathcal{M}}$ in the metamodel $\mathcal{M}$, then the type of n_i is a subtype (direct or indirect) of $n_{\mathcal{M}}$.*
2. *if $\tau_F(e_r)$ is the referential association edge of an association $e_{\mathcal{M}}$ in the metamodel $\mathcal{M}$, we have $(e_{\mathcal{M}}, n_i : n_{\mathcal{M}}) \in I_i$.*

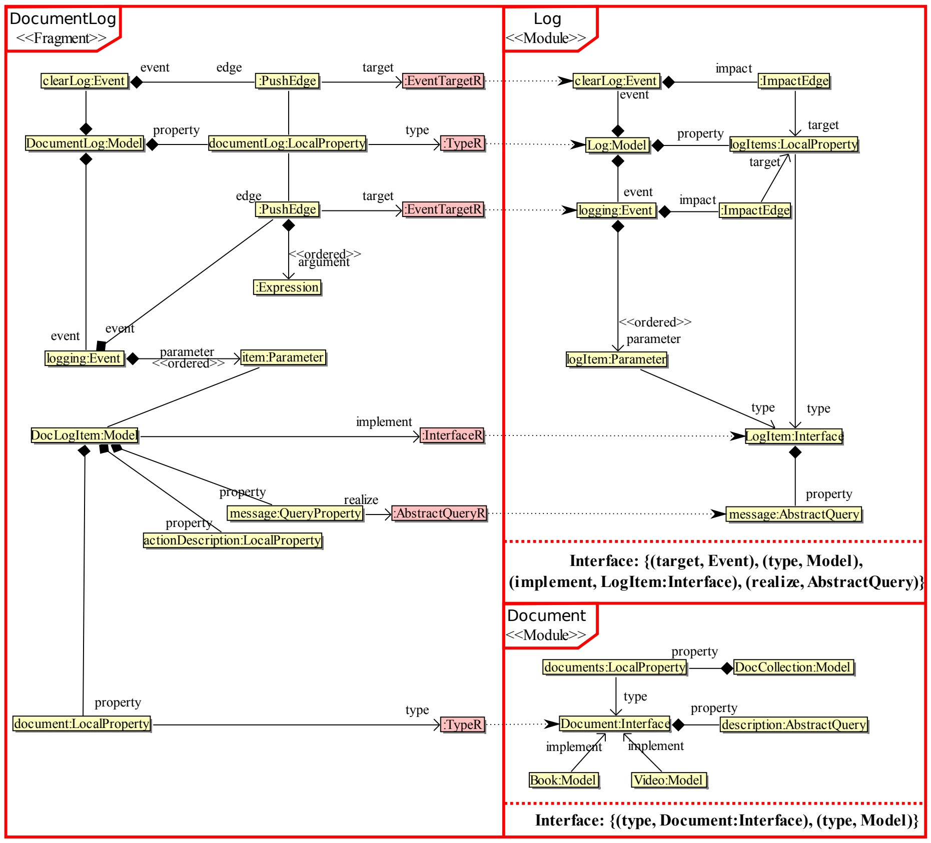

Fig. 4. Model integration of a fragment (left) and two modules (right)

To illustrate this definition consider the fragment *DocumentLog* in figure 4. The integration mapping over modules *Log* and *Document* is indicated by the dotted edges. For instance the referential edge $(: PushEdge, : EventTargetR)$ (at top of figure) is mapped to element *clearLog* of type *Event*. This is consistent with the definition since first *EventTargetR* is a referential class of *EventTarget* and *Event* is a subtype of *EventTarget* and second *(target,Event)* is in the interface of module *Log*.

Definition 10 (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 of* $\mathcal{M}$*.*
4. ρ *is an integration mapping of* F *over module set* $\{M_1 : I_1, \ldots, M_k : I_k\}$*.*

To define the semantics of model integration, we define the result of integrating models $M_1, \ldots, M_k$ conforming to metamodel $\mathcal{M}$ with fragment F to be another model M conforming to the same metamodel $\mathcal{M}$ as the participating models M_i. Informally this result model is defined by identifying (collapsing) the endpoint of a referential edge with the instance which the referential edge is mapped to. To illustrate this, consider the module integration depicted in figure 4: the edge named *target* from $: PushEdge$ to referential node $: EventTargetR$ and the dotted edge from $: EventTargetR$ to $clearLogEvent$ will be collapsed into an edge from $: PushEdge$ to $clearLogEvent$ named *target*. This transformation is repeated for all referential edges leaving the fragment. This is expressed formally in the following definition:

Definition 11 (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 $\tau(v(e_F)) = \mathsf{ori}(\tau_F(e_F))$.

It follows from this definition that the result model of model integration is a well defined model according to Definition 2. The following theorem shows that it is also a valid one, i.e., it conforms to the same metamodel as the models that participate in the module integration.

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. See [9].

7 Integration Hierarchies

By repeatedly applying model integration steps we can build up, starting from a set of disjoint modules, a hierarchy of modules which we now formally define.

Definition 12 (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 a set of modules and has an associated integration mapping over these modules;*
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 model that is derived by integrating the fragment with its successor modules).*

The following theorem holds.

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

Proof. See [9].

The system model is represented 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 that there exists a special empty fragment that integrates all these top models.

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

Definition 13 (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.*

In figure 5 we give an example of an integration hierarchy for a document management system. The three nodes at the bottom represent disjoint models representing the graphical user interface (Gui), the document business domain (Document) and the logging (Log). One level up in the figure the DocumentGui fragment integrates the Gui and Document models: it represents the graphical user interface adapted to managing documents. At the same level the DocumentLog fragment integrates the Document and Log models; it provides logging facilities for the Document model. At the top level the fragment named DMS integrates the lower level models into a complete document management system.

At this point we would like to briefly explain why our approach facilitates model comprehension. First note that the base models (at the lowest level of the integration hierarchy) are self-contained. We can express this using terminology of component-based systems by saying that our models have only an export interface but not an import interface. At higher levels we hook up a fragment to the export interfaces of lower level models. The fragment uses a language (the fragment metamodel) very similar to the metamodel of the participant models. Furthermore the composition mechanism itself is quite simple: it consists in simply identifying referential nodes of the fragment with instances in the interface of the participant modules (as described in the previous section).

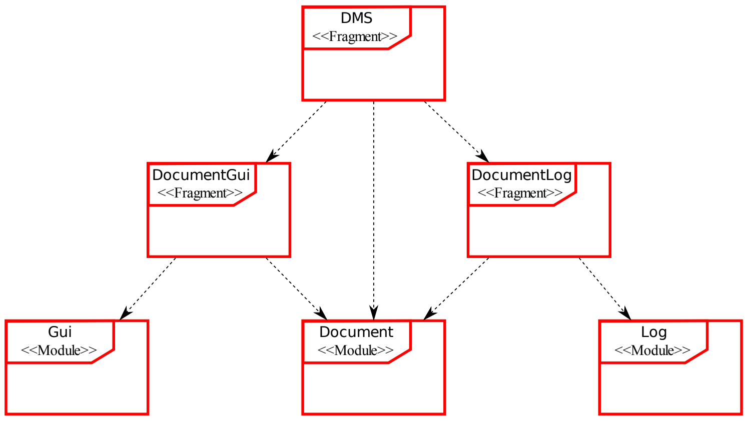

Fig. 5. Model integration of a fragment (left) and two modules (right)

8 Discussion and Related Work

We start by reviewing the main contributions of this paper. These are, as claimed in the introduction, the following features of the composition technique: (a) it is simple, (b) it is modular, (c) it does not require an additional language to express composition, and (d) it is formally defined. As far as simplicity is concerned we note that the input to the composition are the fragment, a set of modules, and a mapping of the referential edges in the fragment to instances of the models. The actual composition (expressed in definition 11) simply collapses the referential edge with the target instance, a straightforward transformation. The modularity of our approach is based on the idea of equipping models with interfaces hiding some of the elements inside the model from the view of the fragment. The resulting concept of a module was presented in section 5. Because of the simplicity of the composition mechanism (described above) we do indeed not require a separate language for expressing the composition. Finally we have given formal definitions of the concepts related to model integration, underlining its formal foundation.

We now consider related work. 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 [2]. The definitions in that paper are based on examining three model composition frameworks: the Glue Generator Tool [2, 3], the Epsilon Merging Language [10], and the Atlas Model Weaver [1]. The model composition techniques expressible in these frameworks may be called white-box model composition techniques: they are usually based on having full access to the modelling elements within each model. They also differ from our approach by allowing composition operations of high complexity to be specified. Of course this also means that these techniques are more expressive than ours:

in particular they do not restrict participating models to conform to the same metamodel.

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) [4].

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 [13] addresses the problem of information hiding at the level of metamodels that are instances of MOF. Besides its focus on metamodels this work differs from our approach in the way it expresses information hiding: it assumes that each metamodel has import and export interfaces. Metamodel composition is expressed by binding elements in import interfaces to elements in export interfaces. In our framework, on the other hand, models have only export interfaces. Composition is realized by combining export interfaces via fragments.

The authors of [5] develop a theory of model interfaces and interface composition in the context of dealing with soft references across XML models, i.e., untyped, string-based references between XML documents. 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 underlying our approach.

9 Conclusion

In this paper we have presented a modular technique for composing models conforming to the same metamodel. It differs from most existing model composition techniques in three important ways: first, information hiding is realized by each model offering an export interface (using terminology from component-based systems) but no import interface. This means that models are really self-contained: they cannot access features that they do not "implement". Second, a separate language for expressing the composition is not required since the composition only requires mapping referential nodes in the fragment to instances in the participating models. Third, for the reason just given the composition itself is quite simple. All of these differences help in reducing the coupling between the participant models and the composed model, thus facilitating the comprehension and maintenance of the composed 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 edges. This will be the subject of future work.

Another line of future investigations concerns the model comprehension aspects of our model composition technique. The benefits for model comprehension are addressed rather summarily in the present paper (at the end of section 7). Future work will address this question in more detail. In particular we are interested in the reverse process of building model hierarchies. Suppose we have an existing model. Can we decompose it into a model hierarchy, thereby facilitating the comprehension of the initial model?

References

1. Atlas model weaver project, 2005. `http://www.eclipse.org/gmt/amw/`.
2. 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.
3. 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.
4. George T. Heineman and William T. Councill. *Component-Based Software Engineering: Putting the Pieces Together (ACM Press)*. Addison-Wesley Professional, 2001.
5. 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.
6. Pierre Kelsen. A declarative executable model for object-based systems based on functional decomposition. In *the Proceedings of 1st International Conference on Software and Data Technologies (ICSOFT 2006)*, pages 63–71, Setúbal, Portugal, 2006.
7. Pierre Kelsen and Qin Ma. A formal definition of the EP language. Technical Report TR-LASSY-08-03, Laboratory for Advanced Software Systems, University of Luxembourg, May 2008. `http://democles.lassy.uni.lu/documentation/TR_LASSY_08_03.pdf`.
8. Pierre Kelsen and Qin Ma. A lightweight approach for defining the formal semantics of a modeling language. In *the Proceedings of ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2008)*, volume LNCS 5301, pages 690–704, 2008.
9. Pierre Kelsen and Qin Ma. A modular model composition technique. Technical Report TR-LASSY-09-01, Laboratory for Advanced Software Systems, University

of Luxembourg, May 2009. `http://democles.lassy.uni.lu/documentation/TR_LASSY_09_01.pdf`.

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. 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.