A Canonical Scheme for Model Composition

Jean Bézivin[1], Salim Bouzitouna[2], Marcos Didonet Del Fabro[1], Marie-Pierre Gervais[2], Fréderic Jouault[1], Dimitrios Kolovos[3], Ivan Kurtev[1], and Richard F. Paige[3]

[1] ATLAS Group (INRIA & LINA), Université de Nantes, France
{Jean.Bezivin, Frederic.Jouault, Marcos.Didonet-Del-Fabro, Ivan.Kurtev}@univ-nantes.fr
[2] Université de Paris-6, France
{Salim.Bouzitouna, Marie-Pierre, Gervais}@lip6.fr
[3] Department of Computer Science, University of York, UK
{paige, dkolovos}@cs.york.ac.uk

Abstract. There is little agreement on terminology in model composition, and even less on key characteristics of a model composition solution. We present three composition frameworks: the Atlas Model Weaver, the Epsilon Merging Language, and the Glue Generator Tool, and from them derive a core set of common definitions. We use this to outline the key requirements of a model composition solution, in terms of language and tool support.

1 Introduction

Model composition involves combining different models in a Model-Driven Development process. Model composition is an emerging research field, based on related work in aspect-oriented modelling [14], database schema integration [11,12], and model transformation [15]. There is not, as yet, an agreed vocabulary, glossary, and set of definitions on model composition. Nor is there an agreed set of basic requirements for model composition languages and tools.

This paper addresses these issues by deriving a common set of definitions for model composition, and from this deriving a set of fundamental requirements for model composition languages and tools. We base our presentation on an assessment of three functional frameworks: the Glue Generator Tool [7, 8], the Epsilon Merging Language [6], and the Atlas Model Weaver [10]. Based on these frameworks, we derive a set of common definitions, before presenting a set of solution requirements.

Let us consider the simplified situation where there are three models Ma, Mx and Mb conforming to metamodels MMa, MMx and MMb. A typical transformation problem may be stated as follows: given Ma and Mx compute Mb. Mx is the transformation model that, when applied to Ma, produces Mb. There are no specific constraints on metamodels MMa and MMb, but metamodel MMx defines the transformation language, for example ATL [15]. Alternatively, computing Mx from Ma and Mb is clearly a more difficult issue, different from a transformation. This situation corresponds to one kind of composition problem which is, in general more complex. In this case we have usually no constraint on metamodels MMa, MMx or MMb. We see here that model transformation is quite well understood while model composition still needs further investigation. Furthermore, a composition may

A. Rensink and J. Warmer (Eds.): ECMDA-FA 2006, LNCS 4066, pp. 346 – 360, 2006.
© Springer-Verlag Berlin Heidelberg 2006

sometimes also be perceived as a transformation with two input models and one output model.

Which kind of composition scheme are we going to use in the aforementioned composition examples? Can the composition of Ma and Mb to produce Mx be completely automated or do we need in some cases to resort to some external inter-vention? Can we define merging heuristics that could be applied to Ma and Mb in order to produce Mx? Should model composition be considered as a one shot operation or could it be decomposed in several phases of discovering correspondences first and then transforming these correspondences into operational mappings that could be solved by multi-input model transformation? There are many open questions in the field of model composition. There are also several partial solutions. What we need is to place these various solutions within one common conceptual framework in order to identify a canonical scheme that will allow us to compare them and to show their complementarities.

This paper is organized as follows. Section 2 describes three model composition solutions that have been independently developed in the context of the ModelWare European Integrated project. These solutions are addressing different goals and may be typical of which kind of problems could be solved by model composition techniques. Section 3 provides a glossary and some common definitions because we recognize that without solid foundations it will not be possible to produce any canonical scheme for model composition. These definitions are based on graph theory. Building on the two previous parts, Section 4 proposes an initial set of requirements for model composition frameworks. While this work does not claim completeness, it concludes that the problem of model composition should not be confused with plain model transformation. The issues are much broader and there is an urgent need for additional work in this field.

2 Model Composition Frameworks

We now describe three functional model composition frameworks, and from these descriptions identify a canonical scheme for model composition based on a glossary and a common set of definitions. A significant summary of the state-of-the-art in model composition would be a useful contribution but goes beyond the scope of this paper. In particular, related work may be found in XML, aspect-oriented programmming, data engineering, the semantic web, and elsewhere.

2.1 Atlas Model Weaver (AMW)

The Atlas Model Weaver is a model composition framework that uses model weaving and model transformations to produce and execute composition operations. The model resulting from a composition may contain parts or all of the elements of the input models, and it may also have new elements. AMW has been used to handle several problems in data engineering [16]. The tool is available as open source from the Eclipse GMT project [10].

Let us illustrate the composition of two simple object models M_A and M_B into a model M_{AB}. M_A contains class *Teacher*. M_B contains classes *Professor* and *AssistantProfessor*. From this example, we illustrate three possibilities (there may be

more) of output model. First, M_{AB} contains one class *Professor* that contains the information from all the other three classes. Second, M_{AB} contains classes *Professor* and *AssistantProfessor*; *Teacher* is combined with *Professor*. Third, M_{AB} has three classes: *Professor*, *AssistantProfessor* as in the previous scenario, and a new class *VisitingProfessor*. This class contains information about occasional visitors.

There are different options to implement a composition operation. One is to write a transformation by hand. However, model composition scenarios have a set of frequently used primitives with specific semantics, such as "merge", "override" or "extends". These primitives link concepts that represent similar information. We must raise the abstraction level of current transformation languages to create composition links. The links must be saved, as they are the specification of the operation.

In AMW, the production of a composition operation is divided in two phases. First, a weaving model captures the links between the input model elements, for example indicating that *Teacher* and *Professor* are combined into *Professor*, or that *Visiting-Professor* is a new class to be created. The weaving model conforms to a weaving metamodel. It is a domain specific metamodel dedicated to composition scenarios. It contains elements such as "rename", "override", "merge", and elements specifying how to solve conflicts between the input models.

Second, the weaving model is used to generate a transformation. This transformation is the final composition operation. The code complexity is not an issue here because the transformation is automatically produced. The transformation takes two input models and produces the composed model as output.

2.1.1 Weaving Model

In order to provide a description of a weaving model, let us suppose we have two metamodels *LeftMM* and *RightMM*. We need to establish links between their elements. The type of links specifies how the elements are composed. Some issues need be considered regarding the set of links between elements of both metamodels:

- The set of links cannot be automatically generated because it is often based on design decisions or various complex heuristics;
- It should be possible to save this set of links as a whole, in order to use them later to produce the composition operation.

The weaving model conforms to a weaving metamodel *WMM*. The weaving model is produced by a match operation. A match operation is a combination of automatic techniques with user interaction. The produced weaving model relates with the source and target metamodels *LeftMM* and *RightMM*. Figure 1 illustrates the conformance relations of *LeftMM*, *RightMM* and *WM*.

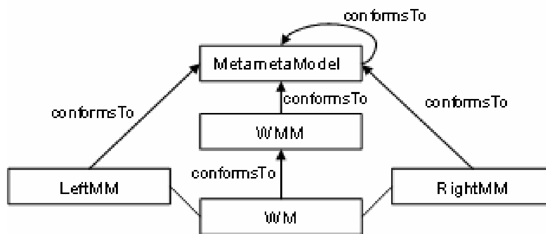

Fig. 1. Weaving conformance relations

Each composition link conforms to *WMM*, which specifies a composition operation. The link types are divided into *matching* and *composition* links. Matching links specify the equivalences between elements. Composition links specify how to solve conflicts and how to compose the related elements, e.g., equivalent elements are merged into one and the right element name is taken as default.

There is no standard weaving metamodel capable of capturing every semantics to compose models. However, various weaving metamodels have a set of common concepts: all provide means to establish links between model elements. We capture this in a *basic weaving metamodel*, and obtain different semantics by extending it. This *extension operation* takes two metamodels as input and returns a weaving metamodel. The output metamodel contains all elements from the input metamodels.

Figure 2 describes our basic weaving metamodel, which contains a *WElement,* the base element from which all the elements inherit. *WModel* is the root element. *WLink* can be extended to define different matching and composition links, and refers to multiple endpoints. *WLinkEnd* indicates the type of elements that are be composed. *WElementRef* has an identifier (ID) that points to the elements of the input models. Each extension of *WElementRef* implements a different identification mechanism, for example XMI-ID. *WModel* also contains *WModelRefs*, which is equivalent to the reference of *WLinkEnd* and *WElementRef*, but for models as a whole.

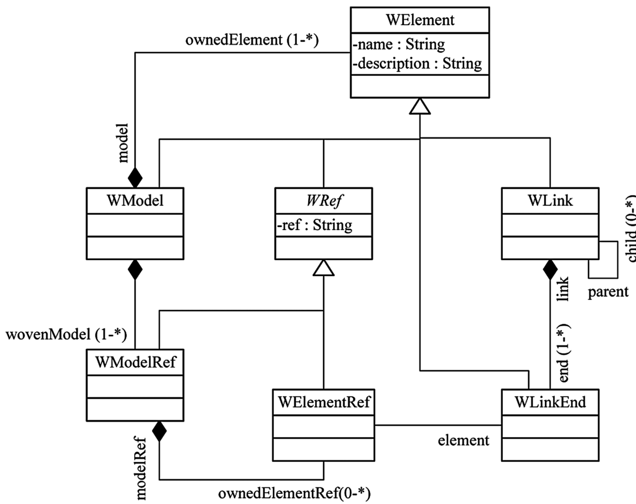

Fig. 2. Basic weaving metamodel

2.1.2 Weaving and Transformations as a Composition Operation

The weaving model is a high-level specification for the composition operation. It is not an executable entity, i.e., there is no specific composition engine to execute it. The composition operation is obtained by translating the weaving model into a transformation model. It is automatically produced by a higher-order transformation (HOT). A HOT is a transformation that either takes a transformation model as input,

either produces a transformation model as output, or both. There is one different HOT for every weaving metamodel, which defines the semantics of the weaving metamodel by transforming its instances to executable transformations.

The elements of the weaving models are transformed into specific composition code patterns. For instance one may define an element called *Union* that combine *Professor* and *AssistantProfessor* into a single element. Every time the weaving model is modified, the composition operation is regenerated. The composition operations are produced for different transformation languages, such as ATL, SQL or XSLT. They are further serialized into the appropriated format (often text). The serialized form takes as input the models to be composed. It executes the composition between the data sources in the dedicated transformation engine.

2.2 Glue Generator Tool

The Glue Generator Tool (GGT) [7, 8] is a framework dedicated to the reuse of existing MDA [1] applications, without alteration, to build new ones. In the MDA approach, this reuse relates to both PIM and PSM reuse, and to composition as well as extension or modification of existing PIMs and PSMs.

The GGT framework comes with:

- a metamodel of composition rules, implemented with EMF. Three categories of rules are proposed [7]. The *correspondence rules* are used to put in correspondence related model elements. The *merge rules* are dedicated to the composition. They identify which model elements from the source models will merge. The *override rules* are dedicated to the modification. They identify which model elements in a source model will be replaced.
- a Glue Generator Tool for EJB 2.0, implemented using an EMF repository.

2.2.1 Scope of Work and Approach

GGT supports construction of a new application from existing ones in the case where the original applications were built using PIMs, PSMs, transformations and code. It also supports functionality extension of applications built using an MDA approach, and in both cases without modifying the original applications.

As a preliminary requirement, GGT considers that the reuse of PIMs must not modify the existing PIMs. Consequently it provides a designer with means to express PIM reuse in terms of model composition, extension, and modification. Currently, PIMs are expressed in UML 2.0 in terms of class and sequence diagrams. This expression of composition (*EC*) is in addition to the existing PIMs to be composed (or extended or modified). Once this expression *EC* is available, GGT provides a means for its automatic translation into its corresponding pieces of model at the PSM level. The result of this translation is called *glue*, since it binds the existing PSMs according to *EC*. As such, the Glue depends not only on the expression *EC* but also on the type of PSMs that it binds.

For the sake of concreteness, we focus on a specific type of PSM in the presentation that follows: Enterprise Java Beans (EJBs).

2.2.2 Glue for EJB PSMs

The authors in GGT defined the concept of Glue for EJB PSMs according to three statements we established from the analysis of the mapping rules of the UML class and sequence diagrams onto EJB platforms:

- R1) a business class maps onto an Entity bean. Its attributes, depending on whether they are persistent or not, map onto persistent or simple fields.
- R2) a process class maps onto a Session bean. Its attributes map onto fields.
- R3) An association or a dependency between classes, depending on the nature of the class (business or process), maps onto EJB Relationships or EJB references between the corresponding beans.

These three statements can be detailed as follows:

S1) Translating EC is mapping the composition and the override rules expressed between PIM elements onto effective merge and replacement of their corresponding EJB PSM elements.

The three rules Ri illustrate how to map PIM elements onto EJB PSM elements, and can be used for the translation of *EC* defined at PIM level by a designer.

S2) The unit of composition or override at PIM level is the class.

At the PIM level, the designer can express some composition rules aimed at merging some PIM elements, such as:

1) The merge of features (attributes/ operations) of different classes into one feature (attribute/ operation);
2) The merge of classes of different packages into one class;
3) The merge of sub-packages of different other packages into one package.

However, since the encapsulation unit of PIM is the class, all these merges consist in merging classes. The merge of the packages consists of merging their corresponding classes. In addition, the merge of attributes or operations must initially deal with the merge of their container, which are classes. This also holds for the override rules, which consist of replacing elements of one PIM by those of another.

S3) The composition or the override of classes is the merge or the replacement of the corresponding beans

When considering the three rules Ri mentioned above, we note that the classes at PIM level map onto beans.

These three statements enable us to define the Glue as a PSM binding entity responsible for the merge or replacement of beans. These two operations and how the Glue achieves them are described in depth in [8].

2.2.3 Architecture of the Tool

The Glue Generator Tool is responsible for automatic generation of the Glue from the expression of composition defined at PIM level by a designer. To this end, it inputs a composition model defining a set of composition rules, and outputs the Glue that will bind the corresponding PSMs. It consists of the Analyzer, the Generator and the Controller (Figure 3).

The *Analyzer* parses the input composition rules to build a merge tree or a replacement tree according the categories of these rules while checking the semantic and syntactic well-formedness of rules. For semantic well-formedness, the authors of GGT defined a set of constraints on the rules and have developed automated constraint checking operations that run at composition model load time in the analyzer.

The *Generator* is the builder of Glue and consequently is specific to the platform on which the applications run. We currently provide a generator for EJB 2.0 platform and another for JMX platform. A generator consists of an API and its implementation. The API should allow Controller to create/read/modify the models based on the specific platform. The *Controller* is the processor of GGT. It manages the generation of Glue using the API of the Generator. It parses the merge tree and triggers the generation of glue according to a generation mechanism. Since the Glue depends on the kind of platforms, the generation of the controller also depends on the platforms.

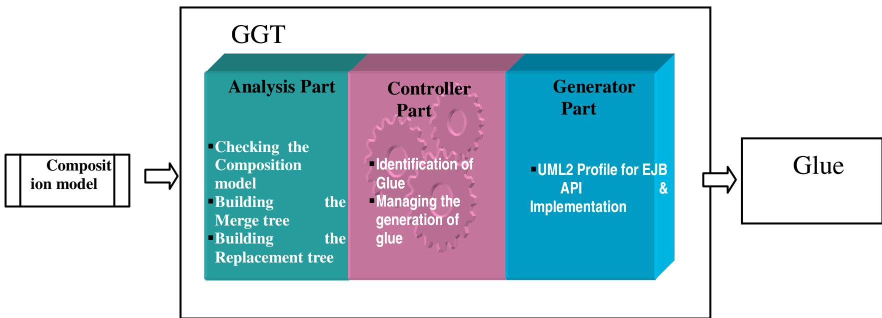

Fig. 3. Architecture of GGT

2.3 Epsilon Merging Language

The Epsilon Merging Language (EML) is a metamodel agnostic language for expressing model compositions. It includes a model comparison and model transformation language as subsets, and is built atop a generic model management language called the Epsilon Object Language (EOL) [6], which is inspired by OCL. An EML specification consists of a set of rules describing how model compositions should be carried out. Rules in EML are of three types: match rules, merge rules, and transform rules. Match rules can be further subdivided into comparison and conformance rules (examples to follow). Each match rule has a unique name and two metaclass names as parameters. The rule itself is composed of a *compare* part and a *conform* part. The rule is executed for all pairs of instances of the metaclasses that appear in the source models. The compare part of a match rule determines whether two instances match, using a minimum set of (syntactic) criteria. The conform part applies only to instances that satisfy the compare part of a rule; the conformance rule part refines this match. If the conformance part of the rule fails, then an exception is raised. An example is shown in Figure 4:

```
abstract rule ModelElements
     match l: Left!ModelElement
     with r: Right!ModelElement
     extends Elements {

     compare {
            return l.name = r.name
            and l.namespace.matches(r.namespace);
     }
}

rule Classes
     match l: Left!Class
     with r: Right!Class
     extends ModelElements {

     conform { return l.isAbstract = r.isAbstract; }
}
```

Fig. 4. Matching rules in EML

The rule on ModelElements is abstract and provides basic behaviour that is used by rules that *extend* it; EML supports rule reuse via inheritance. The behaviour of this abstract rule is to match model elements that have identical names (*l.name=r.name*) and matching namespaces. A similar match rule is used for classes. Classes match when they obey the rules declared in their parent and when the *conform* part of the rule holds, i.e., when classes are either both abstract or both not abstract.

2.3.1 EML Model Element Categorisation

After the execution of all match rules in an EML specification, all model elements are categorised in four groups: those that match and conform; those that match but do not conform; those that do not match; and those to which no match rule has applied (the last category of element produces warnings). The results of this matching process are used in the merging process. In particular, elements that match and conform will be merged with their identified opposite. The specification of merging is captured in *merge rules*. Elements that do not match will be *transformed* into model elements compatible with the target metamodel. This is captured using *transformation rules*.

2.3.2 EML Merge Rules

Merge rules in EML are used to specify the behaviour necessary to compose two instances of model elements that match and conform. Each merge rule consists of a unique name, two metaclass-typed parameters, and a list of the model elements that the rule creates in the target model.

For all pairs of matching instances of the two paramters, the rule is executed and the declared empty model element(s) are created in the target model.. The contents of the newly created model element are defined by the body of the merge rule. Two examples of merge rules are shown in Figure 5:

```
rule ModelElements  {
   merge l: Left!ModelElement
   with r: Right!ModelElement
   into m: Merged!ModelElement


   m.name := l.name;
   m.namespace:=l.namespace.equivalent()
}
```

```
rule Classes {
   merge l: Left!Class
   with r: Right!Class
   into m: Merged!Class
   extends ModelElements

   m.feature := l.feature.
     includeAll(r.feature).
     equivalent();
}
```

Fig. 5. EML Merge Rule

Figure 5 presents two merge rules, one for merging ModelElements and a second for merging UML classes ("Classes"). The first rule applies to all Model Elements and produces a new, merged ModelElement whose name is that of the left original model element, and whose namespace is that of the left original model element. In the second rule, the two parameters, *l:Left!Class* and *r:Right!Class*, are declared; the merge rule is also declared to produce an instance of the Merged!Class metaclass. The "Classes" rule creates a new instance of the Class metaclass, carries out all operations declared in its parent (ModelElements), and sets the feature list of the new class to be the union of all features from the left and right arguments.

There is a slight twist to the merging rule that takes the union of all features from the left and right model elements: the use of the *equivalent()* operation. This operator returns the *equivalent of the model element to which it is applied* in the target model. The equivalent of an element is the result of a merge rule if the element has a matching element in the opposite model; otherwise it is the result of a transform rule. This operator is necessary because the target and source metamodels may differ, and it ensures that all source elements are expressed in the target metamodel.

A key aspect of merging models in EML is that many merge rules can in fact be inferred from the *structure* of the metamodel itself: for example, when merging two classes, a basic merge rule can automatically be inferred that merges the contents of the classes (i.e., behavioural features and attributes). Such inferred rule sets we call *strategies*; further details on them (and on EML) can be found in [6].

3 Glossary and Common Definitions

We propose a set of definitions for a model composition framework. They are an extraction of the common points of AMW, GGT and EML. The formal definitions are intended as a starting point for a common canonical scheme.

The three frameworks follow standards of model-driven development. This means they all have models as the central concept. The models are represented as graphs. In this case it is straightforward to converge to a graph model representation.

Definition 1 (Directed graph). A directed multigraph $G = (N_G, E_G, \Gamma_G)$ consists of a finite set of nodes N_G and a finite set of edges E_G and a mapping function $\Gamma_G : E_G \rightarrow N_G \times N_G$.

Definition 2 (Model). A model $M = (G, \omega, \mu)$ is a triple where:

- $G = (N_G, E_G, \Gamma_G)$ is a directed multigraph,
- ω is itself a model (called the reference model of M) associated to a graph $G_\omega = (N_\omega, E_\omega, \Gamma_\omega)$,
- $\mu : N_G \cup E_G \rightarrow N_\omega$ is a function associating elements (nodes and edges) of G to nodes of G_ω. This means both nodes and edges of G are constrained by nodes from G_ω.

The relation between a model and its reference model is called conformance. This definition allows an indefinite number of levels. However we observe from different domains that usually only three levels are sufficient. We call these three levels metametamodel (M3), metamodel (M2) and terminal model (M1).

We illustrate the three levels with different technical spaces:

- relational database (RDBMS): the instances (M1), the relational schemas (M2) and the relational data model (M3).
- XML: the XML documents (M1), XML schemas (M2) and the XML schema definition (M3).

Definition 3 (Metametamodel). A metametamodel is a model that is its own reference model.

A metametamodel is self-defined. This allows using the same set of composition tools for the three levels in a uniform way.

Definition 4 (Metamodel). A metamodel is a model such that its reference model is a metametamodel.

Definition 5 (Terminal model). A terminal model is a model such that its reference model is a metamodel.

The three approaches provide a way to capture the correspondences between the models to compose. In AMW, the weaving model has matching and composition links. In GGT, the expression of composition (EC) is a model with correspondence, composition and override rules. EML provides comparison rules (ECL) that produce a weaving model that contains the relationships between the model elements. Differently from the previous two approaches, the composition rules are not specified within the same model. They specify merge rules that take as input the result of the comparison rules.

We thus define a correspondence model that captures links between different models. The metamodel of the correspondence model (correspondence metamodel) is extensible, because different matching and composition links are defined (match, override, correspondence, equality, merge, JoinClasses).

Definition 6 (Correspondence model). A correspondence model $C = (G_C, \omega, \mu)$ represents links between elements of different models, such that:

- $S = \{M_i = (G_i, \omega_i, \mu); i = [1..n]\}$ is a set of models,
- G_C has two types of nodes: *links* and *link endpoints*,
- for each *link endpoint*, there is an edge coming from a *link*,

- each *link endpoint* refers to an element e of a model M_i from the set S by the means of identification functions. An identification function ρ takes a *link endpoint* as input and returns an element of a model from the set S.

Consider two models M_A and M_B and a correspondence model C. M_A contains classes *FirstName* and *LastName*, M_B contains class *Name*. The correspondence model C contains three link endpoints; each endpoint refers to elements *FirstName*, *LastName* and *Name*, respectively. There is one link element with outgoing edges to all the three end points.

The correspondence model is created by different procedures. In AMW, the weaving model is created by a user interface and pluggable match algorithms (in Java code). In GGT, the expression of composition is created by a user interface based on EMF. In EML, a match operation is defined using comparison rules (ECL). These rules search for relationships between the models elements. The process of creating the correspondence model is encapsulated in a match operation. The matching rules produce a weaving model as result.

Definition 7 (Match operation). Match is an operation C = Match (S) that takes a set of models $S = \{M_i = (G_i, \omega_i, \mu_i); i = [1..n]\}$ as input, searches for equivalences between their elements and produces a correspondence model C as output.

The match operator does not have fixed semantics. The semantic is defined with comparison and conformance rules. Comparison rules determine syntactic similarities between model elements. Conformance rules identify if a subset of syntactically similar elements are semantically compatible.

In all solutions there are translation and generation procedures. In AMW, transformations are used for executing the composition. The composition operation is generated using HOTs. In GGT, a Glue is automatically produced from the expression of composition. In EML, transformations are used as part of composition rules to add elements that do not match in the input models. This generation procedures are subsumed in the notion of model transformations. AMW uses metamodel extension to extend the basic weaving metamodel before generating a transformation. The definitions of metamodel extension and model transformation are given below.

Definition 8 (Metamodel extension operation). The operation MM_A = Extend (MM_A, MM_B) takes two metamodels $MM_A = (G_A, \omega, \mu)$ and $MM_B = (G_B, \omega, \mu)$ as input and extends G_A with all nodes and edges of G_B. The operation main requirement is to create at least one new edge in the resulting metamodel from an element $m_A \in N_{GA} \cup E_{GA}$ to an element $m_B \in N_{GB} \cup E_{GB}$. We assume that there are no conflicts between the two metamodels.

Consider two class-based metamodels MM_A and MM_B. MM_A contain classes *Person* and *Address*. One person refers to many addresses. MM_B contain classes *Teacher* and *Student*. MM_A is extended with the elements of MM_B. The class *Professor*, classes *Teacher* and *Student* are copied to MM_A, and they is an inheritance relation with *Person*.

Definition 9 (Model transformation). A model transformation is an operation that takes a set of models as input, executes a set of rules over the model(s) elements and produces a set of models as output.

A transformation has the following signature OUT = T(IN) where T is the transformation name, IN is a set of input models and OUT a set of output models. The transformation T translates the input models IN into the output models OUT. A transformation is a model. This means that all general operations on models may be applied to transformations (including transformations).

In AMW, the weaving model is a high-level specification for the composition. It produces a transformation that is the executable composition operation. This transformation receives two or more models as input and produces the composed model as output. In GGT, the compose operation is a Glue. A Glue is a domain specific structure to compose models. A Glue does not create a new composed model, but an intermediary structure (for example a Bean for composing EJBs) that virtually compose two input models. In EML, there are a set of merge rules to execute the composition. Model elements that are not explicitly referenced in the merge rules are composed by the means of merge strategies.

Finally we define the compose operation on two models:

Definition 10 (Compose operation). The compose operation M_{AB} = Compose (M_A, M_B, C_{AB}) takes two models M_A, M_B and a correspondence model C_{AB} between them as input and combines their elements into a new output model.

In the three approaches there are some differences in the terminology to specify what a composition is. Besides composition, the second most employed term is *merge*. However it is advisable to separate merge and composition. Composition is a more general operation. The semantic is specified in the different operations by a set of rules, and it varies from case to case. Merge, however, is a special case of model composition. Merge has information preservation constraints, i.e., all the information from the input models should be present in the output models, and no duplicate information.

Definition 11 (Merge operation). The merge operation M_{AB} = Merge (M_A, M_B, C_{AB}) takes two models M_A, M_B and a correspondence model C_{AB} between them as input, and returns a model M_{AB} including all the information from M_A and M_B, without duplicate information. The correspondence model is created by the match operation. It specifies the elements that are going to be merged.

4 Requirements for Model Composition Frameworks

We now identify a core set of requirements for a model composition framework. By doing so we attempt to complement the canonical definitions for model composition presented in Section 3 with a concrete set of minimal requirements for a model composition framework. Obviously, this is an initial set of requirements and it will likely need refinement after more practical experience and experiments with the frameworks have been carried out.

4.1 Requirements for a Model Composition Framework

A model composition framework *must* provide at least the following operations:

- means to identify *corresponding* elements in the models that are to be composed (e.g., MOF classes with the same MOF identifier may be said to correspond, e.g., a weaving model or a set of rules).

- means to define how corresponding elements are to be *merged* and *composed* in producing the target model;
- means to define how elements that do not correspond can be *transformed* to the target metamodel, in order to, e.g., not lose information.
- means to manage and *reuse* correspondences, merges, and compose operations. In AMW this is supported via metamodel extension (e.g., by extending a weaving model), whereas in EML this is supported via rule inheritance.

Thus, a model composition framework should also provide the means to carry out transformations (e.g., via MOF 2.0 QVT or ATL) to satisfy the fourth requirement. In order to satisfy the first two requirements, a model composition framework should include the means to *compare models*.

Two *desirable*, practical requirements can be identified from the previous sections:

- A model composition framework should provide the means for minimising the effort expended by the developer to write composition or merge operations, e.g., by allowing rules to be inferred by metamodel structure (e.g., merging strategies in EML) or by allowing expressions of composition or weaving models to be reused.
- A model composition framework should be metamodel independent to support backwards compatibility, future extension, and a wide suite of modelling tools.

4.2 Requirements on Model Composition Tools

Tool support for model composition must provide at least the following:

- validation and verification of model composition operations, i.e., syntax and type checking of rules, merging models, etc.
- a virtual machine (or similar means) for executing composition operations;
- a debugger, for analysing failures and inconsistencies that arise during the composition process
- a serialisation mechanism for loading and saving models.

4.3 Comparison of AMW, GGT, and EML

We summarise the three previously described model composition frameworks against the requirements identified in Section 4.1 and 4.2. The results of the comparison are in Table 1; columns represent a particular framework, whereas rows represent a model composition framework operation or feature. We note that all three frameworks provide reasonably comprehensive coverage of tool requirements (though only AMW provides debugging support via its integration with ATL).

We can observe from the summary in Table 1 that already we are seeing a convergence of functionality in several of the existing frameworks: all three frameworks support most, if not all of the operations described in the canonical set of definitions, and it is already possible to loosely couple some of the frameworks (AMW and EML) together via weaving models.

Table 1. Comparison of model composition frameworks

	GGT	**AMW**	**EML**
Compose	Glue	Weaving model	Merge rules
Merge	Glue	Weaving model	Merge rules that are information lossless.
Transform	Automatically carried out.	ATL transformations	Transform rules
Match	Expression of correspondence (EC), via EMF GUI	Weaving model via EMF GUI	Comparison rules which produce weaving models.
Correspondence	EC	Weaving model	Comparison rules
Metamodel extn.	No	Yes	Indirectly, via generation of weaving model imported by AMW.
Tool support	No debugger.	All.	No debugger.

5 Conclusions

The main contributions of this paper are a canonical set of definitions regarding model composition, and set of requirements for model composition frameworks. The intent is that the canonical scheme, definitions, and requirements will be helpful for comparing different model composition solutions, building new solutions, and assessing the completeness and coherency of existing solutions. The contributions of this paper may also be helpful in any future standardisation efforts – within or without of the OMG – on model composition. We expect to work further on more closely aligning the three frameworks described in this paper, and to explore additional operations that engineers find helpful in model composition scenarios.

The fact that three different solutions for model composition have been developed in the same project is not the mere result of hazard. It shows that the problem is of practical importance and takes multiple forms. There is an obvious need for unification and conceptualization in the field. As discussed in [15], the QVT OMG model transformation proposal [2] only marginally addresses the composition issues. What we have done in this paper is to gather some experimental material that may help giving first class status to model composition as has been done previously with model transformation techniques.

Acknowledgement

The work in this paper was supported by the European Commission via the MODELWARE project. The MODELWARE project is co-funded by the European Commission under the “Information Society Technologies” Sixth Framework

Programme (2002-2006). Information included in this document reflects only the authors' views. The European Commission is not liable for any use that may be made of the information contained herein.

References

1. Object Management Group. Model Driven Architecture official web-site. Internet resource. http://www.omg.org/mda/.
2. Meta Object Facility Queries-Views-Transformations. Internet resource. http:// neptune. irit.fr/ Biblio/ qvt specification.shtml.
3. Object Management Group. XMI specification. Internet resource. http://www.omg.org/ technology/documents/formal/xmi.htm.
4. Object Management Group. Meta Object Facility official web-site. Internet resource. http://www.omg.org/mof/.
5. Modelware IST Project. Internet resource. http://www.modelware-ist.org.
6. D.S. Kolovos, Epsilon Project Page, http://www.cs.york.ac.uk/~dkolovos
7. S. Bouzitouna and M. P. Gervais, *Composition rules for PIM reuse*, Proceedings of the Second European Workshop on Model Driven Architecture with Emphasis on Methodologies and Transformations (EWMDA'04), Canterbury, UK, September 2004, pp36-43
8. S. Bouzitouna, M. P. Gervais and X. Blanc, *Model Reuse in MDA*, Proceedings of the International Conference on Software Engineering Research and Practice (SERP'05), Las Vegas, USA, June 2005.
9. M Lenzerini Data integration: a theoretical perspective, Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of database systems, June 03-05, 2002, Madison, Wisconsin
10. Atlas Model Weaver Project Web Page. http://www.eclipse.org/gmt/amw/, 2005.
11. R. Pottinger and P. Bernstein. Towards Model Composition, in *Proc. VLDB 2003*, ACM, 2003.
12. C. Batini and M. Lenzerini. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys* 18(4), December 1986.
13. R. Reddy, R. France, S. Ghosh, F. Fleurey, B. Baudry. Model Composition: a Signature Based Approach. In *Proc. Workshop on Aspect-Oriented Modelling,* co-located with MODELS 2005, October 2005.
14. T. Cottenier, A. van den Berg and T. Elrad. Modelling Aspect-Oriented Compositions. In *Proc. Workshop on Aspect-Oriented Modelling,* co-located with MODELS 2005, October 2005.
15. F. Jouault and I. Kurtev. On the Architectural Alignment of ATL and QVT. *Proc. Symposium on Applied Computing (SAC 06)*, ACM Press, April 2006.
16. M. Didonet Del Fabro, J. Bézivin,, F. Jouault, and P. Valduriez. Applying Generic Model Management to Data Mapping. *Proc. Journées Bases de Données Avancées (BDA05)*, Saint Malo, France.