A Generic Traceability Framework for Model Composition Operation

Youness Laghouaouta, Adil Anwar, Mahmoud Nassar, Jean-Michel Bruel

A Generic Traceability Framework for Model Composition Operation

Youness Laghouaouta[1], Adil Anwar[2] Mahmoud Nassar[1], and Jean-Michel Bruel[3]

[1] IMS-SIME, ENSIAS, Mohammed V University, Rabat, Morocco
y.laghouaouta@um5s.net.ma,nassar@ensias.ma
[2] Siweb, EMI, Mohammed V University, Rabat, Morocco
anwar@emi.ac.ma
[3] IRIT/University of Toulouse, France
bruel@irit.fr

Abstract. In order to handle complexity, model driven engineering aims at building systems by developing several models, where each model represents a specific concern of the system. In this context, designers need mechanisms to validate, synchronize and understand interactions between those perspectives. Model composition deals with these issues but remains a complex task. For these reasons, we believe that a strong traceability mechanism is a key factor to handle relationships between models and manage the complexity of the composition operation. This paper describes a generic approach to keep track of the model composition operation. We also define a traces generation process to adapt our proposal to any specific composition language. Finally, an example is presented to illustrate our contributions.

Key words: model traceability, model composition, aspect-oriented modeling, graph transformations

1 Introduction

Model Driven Engineering (MDE) proposes models to represent all artifacts handled by a software development process. The principle is to raise the abstraction level by using models as first class entities in dedicated model management operations. Besides, systems are built based on various models that express particular viewpoints. This allows managing the system's complexity, but requires mechanisms to validate, synchronize and understand the interrelation between contributing models. The generation of models that cross separate views tackles these tasks.

Nevertheless, the composition of models remains a complex activity. We advocate that traceability mechanisms are key features to handle this complexity. Indeed, traceability information exposes the effects of executing this operation and helps to comprehend relationships existing among managed models. This kind of information supports validation tasks and offers a way to optimize the co-evolution of models and composition chains.

In previous papers [1][2], we treat the tracing of composition specifications written in the Epsilon Merging Language EML [3] and the ATLAS Transformation Language ATL [4]. The current extends these works and focuses on the generic nature of our proposal. Indeed, we aim to keep track of the composition effects regardless of its specification features. For this purpose, we specify a generic traceability framework that allows tracing the model composition operation, and provide a traces generation process for adapting our proposal to a specific composition language.

The remainder of this paper is structured as follows. Section 2 presents an example that motivates the need for tracing the model composition operation. Section 3 provides an overview of the traces generation process. Section 4 details the structuring and the generation of trace links. In Section 5, we demonstrate the soundness of our approach using a specialization case with the dedicated merging language EML [3]. In Section 6, we review the related approaches. Finally, Section 7 summaries this paper and presents future work.

2 Motivating example

We illustrate the necessity to automatically keep track of the model composition operation using a Library Management System (LMS). The composition scenario we have chosen is the merging of structural models that express an extract of the librarian and head librarian activities. Fig. 1(a) shows an excerpt of the class diagram related to the head librarian, while Fig. 1(b) depicts the class diagram that models the specific librarian requirements. As a result of composing these models, we obtained the class diagram depicted in Fig. 1(c). The composed model contains two categories of elements: elements that originate from only one input model (e.g., the class named *Loan*) and those that existed in both of them (e.g., the class named *Book*). Therefore, two kinds of traceability links have to be captured: merging links and translation links. Essentially, this kind of information reveals the logical relation existing among the managed models and specifies how source models contribute to the production of the composed one. Fig. 1 depicts also extracts of traces that have to be captured. Trace links are represented by nodes labeled with 'M' for merging links and 'T' for translation links, while the dashed lines represent the left, right and target references. In the context of model composition, such information has many possible reuses:

- *Validation of the composition*: trace links provide a detailed view of the flow of execution. Indeed, they represent relationships between source model elements and their target equivalents. Through these links, we can verify the consistency and the completeness of the model composition.
- *Support the co-evolution of models*: those links are useful to analyze the impact of changing sub-models during the evolution of the system. For instance, as the class *Loan* is connected by a translation link with the class *Loan* in the merged model, adding a new attribute to the source class will result in adding this attribute to the target one without reestablishing matching links between the source models elements.

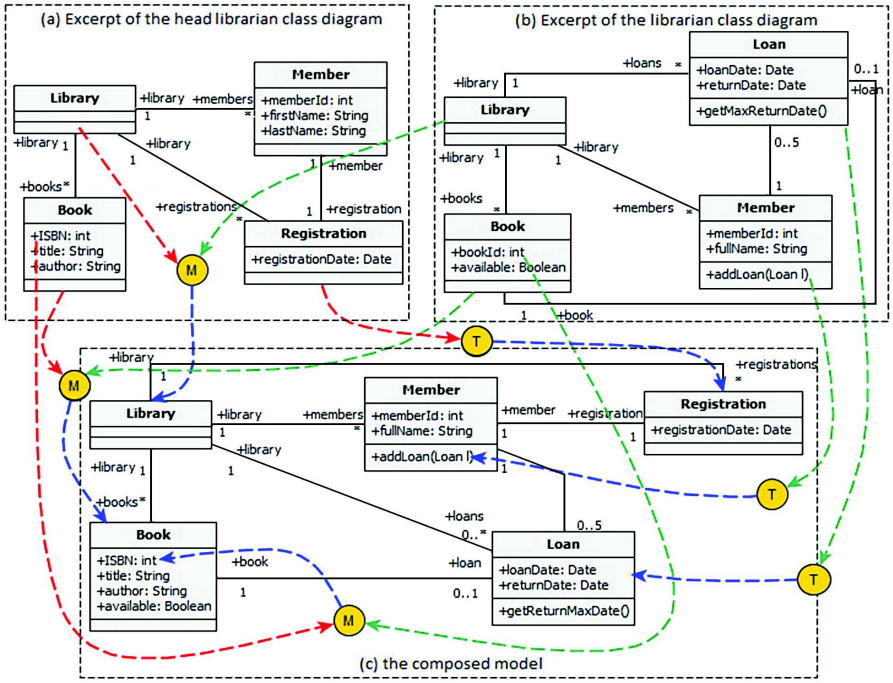

Fig. 1. Motivating example

– *Composition chain optimization*: as a part of a model composition chain, the restriction to source artifacts of a given stage of the overall chain confuses its management. Hence, the use of the trace model can broaden its scope, through the reuse of previous valuable links. As an example, the *bookId* and *ISBN* were considered as describing the same concept. This design decision has been held with the merging link that connects the *ISBN* and *bookId* attributes in the source models with the *ISBN* attribute in the resulting one. Consequently, exploiting this link in subsequent compositions will assist the matching step by specifying that the *ISBN* and *bookId* attribute's names are equivalent.

3 Overview of the traces generation process

In this section, we provide an overview of the way traces are captured and structured. Drawing on the work presented by Jouault [5], we propose to generate the trace model as an additional target model of the specification to trace. However, rather than using a higher-order transformation (HOT) to allow generating traces, we opted for aspect orientation and consider this concern as being a cross-cutting concern. Indeed, the weaving of the traces generation patterns is performed with an Aspect Oriented Modeling (AOM) [6] approach. Traces thus

generated will conform to a generic traceability metamodel presented in Section 4.1.

The weaving process has been defined in such a way that allows the insertion of the traces generation patterns for any composition language while abstracting the concrete specification nature (textual, model-based or graph-based). This solution is organized around two parts (see Fig. 2): the bottom part which is generic and reusable and the top part which is specific for a given composition language. In dealing with the generation task, we simulate the aspect modeling orientation with graph transformations [7]. For this reason, it is necessary to implement a serialization service that provides language specific utilities to parse the concrete specification into the corresponding model (M1).

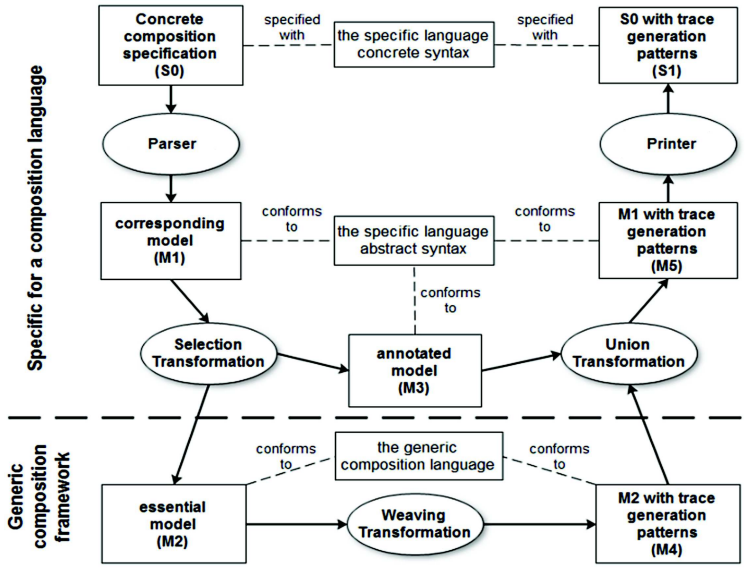

Fig. 2. The traceability weaving process

Besides, in order to specify the graph transformations which implement the weaving mechanism independently from a given composition language, we propose a generic composition language (c.f., Section 4.2). Essentially, this formalization describes the core elements with respect to the traceability perspective and does not take account of all the operational semantic of a composition framework. Accordingly, we have to select the relevant elements from the corresponding model (M1) and therefore we translate them to conform to our generic proposal. This extraction is performed using a specific graph transformation called *selection transformation*. Besides, the specific composition metamodel has to be augmented with traceability capabilities to allow injecting the omitted elements after performing the weaving. Indeed, we add the meta-class *TracedElement* that generalizes all concepts described in the host language. *TracedElement* contains

two attributes: *traceID* to identify each element and *status* to precise whether the element has been selected or omitted. Hence, the *selection transformation* generates two models: the essential model (M2) containing the traceability relevant elements and the model annotated with the aforementioned markers (M3).

Once the essential model is generated, we apply the *weaving transformation* that inserts the traces generation patterns. Thereafter, the *union transformation* transforms the resulting model (M4) to conform to the specific language. On the other hand, it reuses the annotated model (M3) to weave the omitted elements at their relevant containers with respect to the presented markers. Finally, we reproduce the concrete specification which involves the traces generation patterns by using a specific language printer.

4 A generic framework for model composition traceability management

In this section, we detail each component of the generic part of the traces generation process. The traceability framework is based on a generic traceability metamodel accounting for structuring traces. Whereas, the traces generation concern is encapsulated on a traceability aspect which is defined around a generic metamodel that formalizes the core elements of a composition language.

4.1 A generic metamodel for structuring traces

In the literature, several metamodels for the model transformation traceability have been proposed [5][8][9]. The core concept in all of them is the trace link construct, which represents a relationship between a set of source elements and the targets ones. In our context, two categories of relationships have to be expressed: merging links and translation links. However, metamodels addressing the model transformation traceability do not support this case of categorization or poorly express it through the assignment of additional information. In fact, the way trace links are structured must take into account the composition mechanism.

Fig. 3 depicts our composition traceability metamodel [2]. It specifies two types of trace links: merging links and translation links. On the one hand, this categorization allows expressing the composition relationships kinds in a trivial manner. On the other hand, it guides the reuse of traces (e.g., matching correspondences can be deduced from merging links). A merging link connects the source artifacts (belonging to the left and right models) to their target equivalent. While a translation link represents a transition from a left or right element to the target one.

We represent rule invocation by a nesting of trace links which is expressed through parent-child relations among them. This structure provides a multi-scales character to the generated traces and allows the final user to configure the granularity level he desires. In addition, the *Context* concept brings another configuration mechanism. It provides us with the support to represent semantically

rich traces by assigning further information to a subset of trace links. This configuration can be achieved through the definition of relevant expressiveness data (e.g., the composition rule name, the traceability intention ...), where a context attribute is tied to the additional information to be appended to a specific set of traces, and a context is as a well thought out combination of attributes.

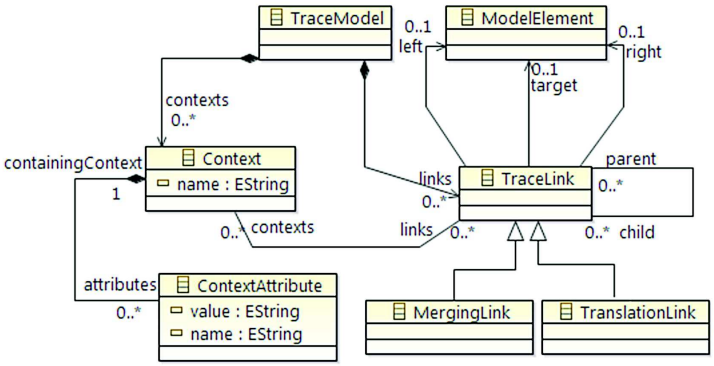

Fig. 3. The composition traceability metamodel

4.2 A generic model composition language

Several approaches addressing the model composition field have been proposed: AMW [10], EML [3], Kompose [11]. In order to provide a generic traceability solution, we have to abstract the composition operation from its syntax nature (textual or graphical) or the language used to express it. A typical composition process involves two major steps: matching and merging [12]. During the first step, correspondences between left and right model elements are calculated. Matching elements are merged while other elements may be transformed into target model elements. In addition, Bezivin *et al.* [12] set a list of requirements for model composition frameworks. We consider these aspects to identify the main elements that constitute the backbone of such an operation (see Fig. 4).

- *Source and target models*: the composition operator combines source models in order to produce the target ones. Hence, the composition specification has to be aware of the relevant information (models name, their metamodel...).
- *Merging rules*: they describe the behavior needed to combine two elements that match with respect to some correspondences criteria. These rules have a name, a statements block which specifies the merging mechanism, and a set of parameters referencing the contributing elements.
- *Translation rules*: their structure is similar to merging rules. However, they are applied to transform elements that have not been merged into the target model, and therefore have at most one source parameter.

– *Rule invocation*: without an adequate mechanism to call rules, the target model looks fragmented. This mechanism enables the weaving of structural relations between target elements (e.g., an attribute belonging to its class) by linking the result of a rule application to the elements previously created. We encapsulate this behavior in an abstract operation named *targetEquivalent*. It resolves the target element corresponding to a source one.

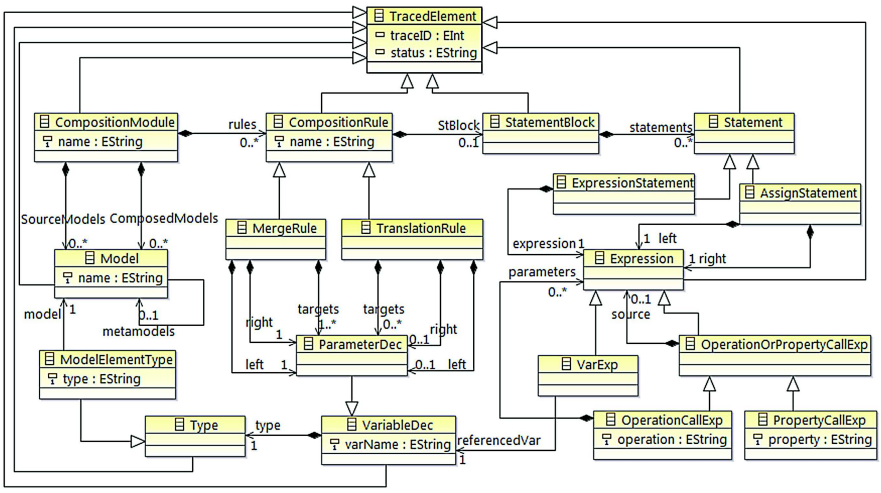

Fig. 4. Abstract syntax for a generic composition language

We formalize these requirements with the metamodel depicted in Fig. 4. Note that this formalization includes a set of concepts defined in the rule based languages EML [3] and ATL [4]. Furthermore, this metamodel does not structure all the operational part of a composition language. It is just a formalization that allows specifying the traceability aspect in a generic way. Essentially, it focuses on the core concepts with regards to the traceability perspective (managed models, merging rules, translation rules and rules invocation) and is restricted to elements that are used to specify the traces generation weaving, such as:

– *Types*: the *Type* meta-class is the basis of all types. It generalizes the *ModelElementType* meta-class which represents a meta-level classifier.
– *Expressions*: it provides expressions to navigate properties (*PropertyCallExp*) and invoke operations (*OperationCallExp*). The *VarExp* expression allows accessing a declared variable.
– *Statements*: the *AssignStatement* elements are used to assign the right value to the left expression, while an expression statement refers to one expression.

The section that follows is concerned implementing the weaving transformations, and clarifying the missing parts of this abstract syntax.

4.3 Graph transformations for traceability weaving

The AOM focuses on modularizing and composing crosscutting concerns during the design phase of a software system. Indeed, both the aspects that encapsulate the crosscutting structure and the base model they crosscut are models [6]. Our objective is to build the trace model without manually encoding the generation patterns and regardless of the specification nature; this approach aligns perfectly with these tasks. It allows encapsulating these patterns in an aspect and abstracts the composition specification through the corresponding model.

Nevertheless, the AOM is a paradigm for conceptualizing aspects, which requires an implementation mechanism. We use graph transformation rules to simulate the aspect orientation. A graph rewriting rule consists of two parts: a left-hand side (LHS) and a right-hand side (RHS). A rule is applied by substituting the objects of the LHS with the objects of the RHS, only if the pattern of the LHS can be matched to a given graph [13]. Therefore, the LHS part is used to determine where the aspect should be applied (the pointcuts); whereas the RHS defines the crosscutting structure that replaces those points (the advice).

We employ a graph transformation unit to weave the traces generation patterns. Its first rule declares the trace model to be an additional target model of the composition to trace. Thereafter, it calls two loop sub-units to trace all the merging and translation rules. Finally, trace links are nested by applying a responsible rule. In what follows, we use the Henshin project [14] to express these graph transformations. Note that the Henshin representation of a rule does not explicit the description of the left and right hand sides. It is based on stereotyping edges to depict the rule application semantic instead.

Trace model declaration We propose to generate the trace model like any other target model. For this purpose, we create two *Model* instances. The first is connected as a new target of the composition and references the trace model, while the second node corresponds to our generic traceability metamodel.

Trace a merge rule Keeping track of merge rules consists of declaring the traceability link, which captures the relationship between the matched source elements and the target one, as being an extra output. For that, the rule depicted in Fig. 5 looks for a *MergeRule* node in the graph corresponding to the specification we wish to trace; subsequently, it appends a new *ParamDec* node of type *MergingLink* to the rule that have been matched as one its target parameters. The added parameter allows the generation of trace links while producing the target elements. Furthermore, this rule creates three assign statements referenced by the *AssignStatement* nodes. Each affects the reference of the corresponding element to the appropriate trace link property (*left*, *right*, and *target*).

Trace a translation rule The application of the rule shown in Fig. 6 inserts the pattern that keeps track of the transition from a source element to its target equivalent. This rule searches for a *TranslationRule* node and declares a new parameter of type *TranslationLink*. This link captures the correspondence

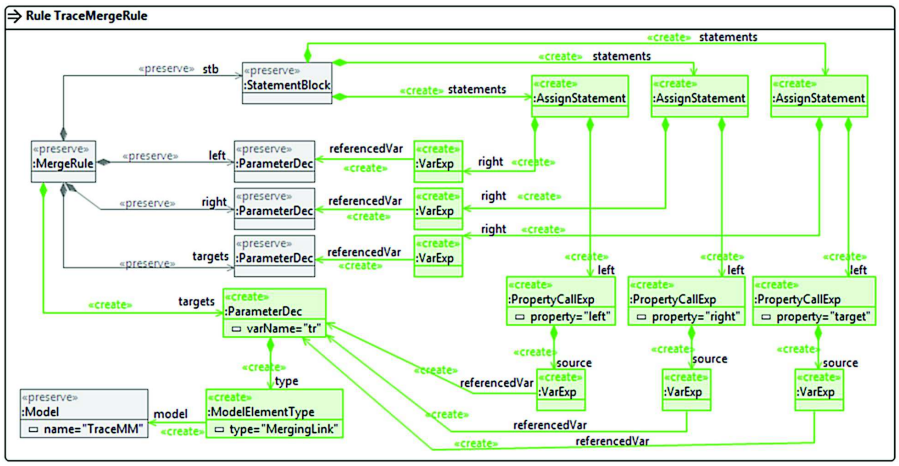

Fig. 5. Trace a merging rule

between the managed source and target elements that are matched with the *ParamDec* nodes stereotyped with *preserve*. As with merging rules, we assign the traceability data to the generated link using *AssignStatement* nodes.

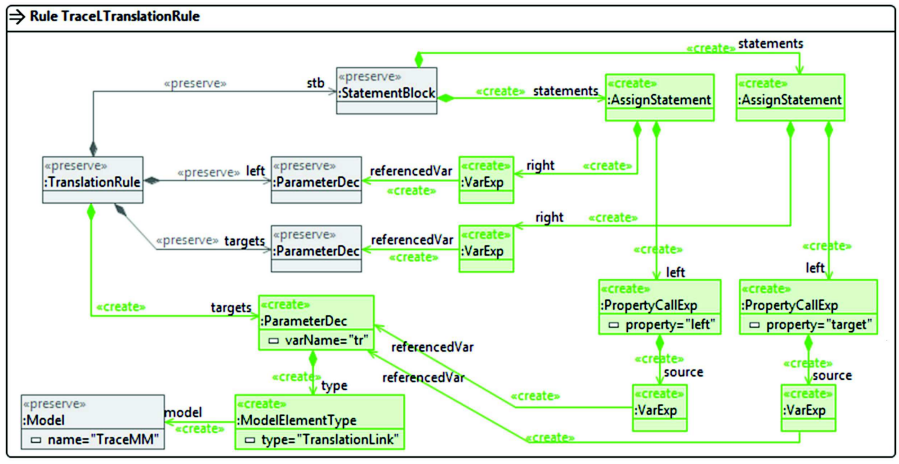

Fig. 6. Trace a translation rule

Trace links nesting In Section 4.2, we discussed the use of rules invocation to structure the composed model. The operation *targetEquivalent* encapsulates this behavior by providing a mechanism for resolving the corresponding target element of a source one. This target equivalent is produced by an anterior application of a given composition rule. We propose to reuse a similar mechanism for

structuring trace links. Actually, it follows from the application of the preceding graph transformations, the production of additional elements corresponding to the traceability links. Therefore, the *targetEquivalent* operation resolves these links as potential equivalents.

Accordingly, we have defined two other abstract operations: *traceEquivalent* and *defaultTargetEquivalent*. They provide a filtering mechanism to select trace links from other target elements. Hence, we can assign the resolved trace equivalent to be a child of the link produced by the current rule (which calls the *targetEquivalent* operation).

Fig. 7 depicts the rule that implements this nesting mechanism. It matches a composition rule involving an invocation of the *targetEquivalent* operation (referenced by the *OperationCallExp* node stereotyped with *delete*). Thereafter, it copies the reference of the element to resolve its target equivalent (which corresponds to the parameter of the *targetEquivalent* call) to the *resolvedElt* variable. Finally, the two other *Statement* nodes allow copying the original call of *targetEquivalent* (using the *defaultTargetEquivalent* operation) and binding the traceability element as a parent of the trace equivalent.

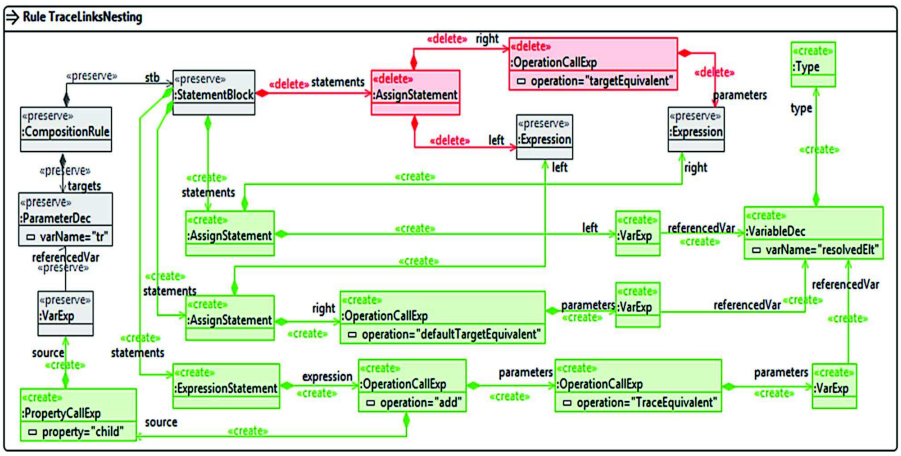

Fig. 7. Trace links nesting rule

5 Specializing the generic framework for a dedicated merging language

In this section, we illustrate the specialization of our generic framework by tracing the merging scenario we have presented before as a motivating example (c.f., Section 2). The composition specification is specified by EML [3], which is a dedicated merging language. We aim through this language enforcing our generation process to reveal the generic character of our traceability framework.

The specialization work consists of identifying the meaning of the main composition elements (managed models, merging rule, translation rule and rule invocation) regarding a given composition approach (e.g., how the rule invocation is performed in EML?). Thereafter, we establish correspondences between the specific representation of each concept and its equivalent conforming to our generic composition language. Those correspondences underpin the selection and the union transformations of the traces generation process (c.f., Section 3).

5.1 Perform the merging scenario with EML

An EML specification is defined using three types of rules: match rules, merge rules, and transform rules. Match rules are applied on the source models to calculate correspondences between their elements. Subsequently, merge rules are used to combine elements that describe the same concept, while transformation rules allow transforming elements that have no corresponding element.

In order to apply our traceability weaving process, we have implemented an EMFText[1] parser that transforms the textual representation of any EML specification into a corresponding model that conforms to the EML abstract syntax[2]. Once the corresponding EML model is generated, the selection transformation is applied to annotate it with the *traceID* and *status* markers and produce the essential model. Table 1 summarizes the mapping between the main generic composition elements and the relevant concept in EML.

Table 1. The EML relevant concepts of the main generic composition elements

Generic concept	**EML relevant concept**
Composition module	Composition module element
Merging rule	Merge rule
Translation rule	Transformation rule
Rule invocation	A call of the *equivalent* operation

Fig. 8 depicts the graph transformation rule that allows selecting merging rules. It looks for a *MergeRule* node (belonging to the EML abstract syntax) with three *ParameterDeclaration* nodes which reference, respectively, the *left*, *right* and *target* elements. Thereafter, it creates the corresponding *MergeRule* node (belonging to our generic composition metamodel) with its connected elements. Note that this rule has two other effects. It duplicates the *traceID* value of the selected elements to the created ones. Also, it changes the value of the *status* attribute into *Selected* in order to annotate the corresponding model.

The weaving transformation is applied on the essential model for weaving the traces generation patterns. Subsequently, the union transformation reinserts the omitted elements into their corresponding containers (i.e., each omitted assign

[1] See `http://www.emftext.org`.
[2] See `http://www.eclipse.org/epsilon/doc/book/`.

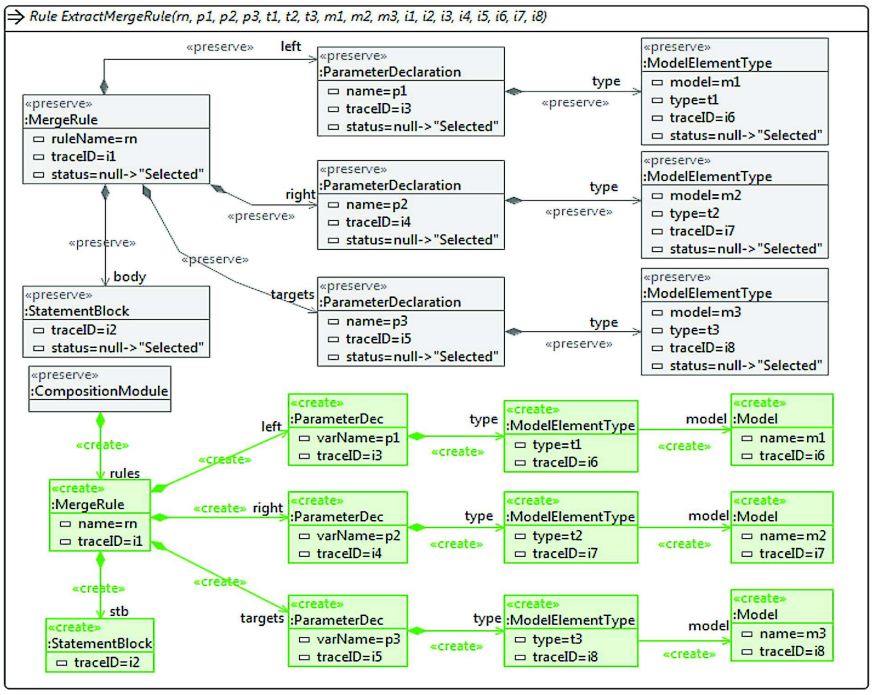

Fig. 8. Selection of merging rules in EML

statement must be nested on its corresponding statements block). The resolution of the relevant container is based on the *tracedID* attribute. Listing 1 depicts the EML rule that merges two matching classes, while Listing 2 represents the resulting modifications over this rule that generate traces.

Listing 1. EML rule to merge two classes

```
1 rule MergeClassWithClass
2 merge l : left!Class
3 with r : right!Class
4 into t : target!Class
5 {
6 t.name = l.name;
7 t.ownedAttribute = l.ownedAttribute.includingAll(r.ownedAttribute).equivalent();
8 }
```

As a result of applying the traces generation weaving process, the traceability parameter is declared as another target parameter and the traceability information is assigned to it (Listing 2: lines 8,11-13). In addition, the call of the *equivalent* operation (Listing 1: line 7) has been captured and replaced with the fragment that allows to nest traces (Listing 2: lines 14-16). Note that the generic operations *defaultTargetEquivalent* and *traceEquivalent* was translated into the host language using the EML *select* operation. Besides, the assignment of the *name* property value (Listing 1: line 6) was marked as omitted and has been injected in the resulting specification (Listing 2: line 10).

Listing 2. EML rule to merge two classes with traces generation

```
pre
{
var resolvedElt : new Any ;
}
rule MergeClassWithClass
merge l : left!Class
with r : right!Class
into t : target!Class , tr:trace!MergingLink
{
t.name = l.name;
tr.left=l;
tr.right=r;
tr.target=t;
resolvedElt = l.ownedAttribute.includingAll(r.ownedAttribute);
t.ownedAttribute = element.equivalent().select(it | not it.isKindOf(trace!TraceLink));
tr.child.add(element.equivalent().select(it | it.isKindOf(trace!TraceLink)));
}
```

5.2 Results

In Fig. 9 we provide an extract of the trace model generated with our framework. Note that we have used the Emf2gv[3] project to provide a user friendly representation of traces. The trace model conforms to our composition traceability metamodel and captures relationships between the contributing models and the merged one. The dashed lines represent the left, right and target references; while the trace links nesting is represented with solid lines.

The trace model contains two types of trace links that are generated with respect to the composition relationships kinds. For instance, the first level merging link connects the composed model to the *Librarian* and *Head Librarian* class diagrams. The contained merging link connects the *Book* classes corresponding, respectively, to the librarian and head librarian concerns, and the composed model. Fig. 9 depicts also a translation link that represents the transition from the *Loan* class in the *Librarian* class diagram to its target equivalent. Furthermore, the nesting of traces is closely modeled on the rule invocation sequence.

6 Related work

In the literature, several works dealing the traceability of model driven development operations are presented [5][8][9][15]. We focus here on two of them that are distinguished by their generic nature:

Grammel and Kastenholz [9] have defined a generic approach to trace various model transformation approaches. Their proposal is based on a generic interface that offers two mechanisms for augmenting a transformation engine with traceability. The first one consists of transforming the implicit trace model to conform to their generic traceability metamodel. Besides, in the case of a lack of an implicit traceability tool, they provide support to generate traces based on Aspect Oriented Programming (AOP).

[3] See `http://sourceforge.net/projects/emf2gv`.

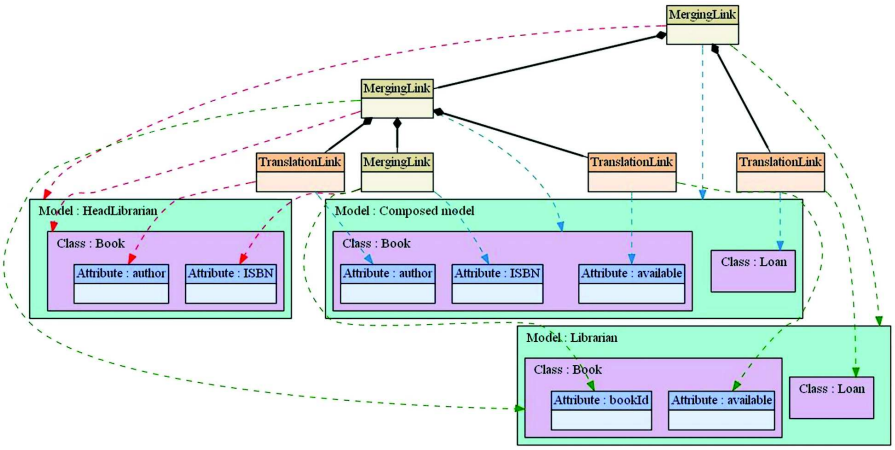

Fig. 9. Extract of the generated trace model

In [15], Vara *et al.* propose a methodological framework that supports the model-driven development of model transformations and allows generating traces freely as a side effect. The extraction of traceability relationships is implemented by a HOT operating on the high-level specification of transformation. This specification is augmented with the traces generation patterns and a set of transformations are described for producing the lower-level transformation models. Thus, the executable transformation generates the target models accompanied by an additional model capturing traceability links. However, the applicability of the approach is restricted to newly model transformations developed by this framework, and no mechanism is defined to trace pre-existing transformations.

Our approach makes use of the benefits of the presented works while focusing on model composition. As for the approach of Grammel and Kastenholz [9], we adopt aspect orientation to generate traces. However, instead of using AOP, we have chosen an AOM solution to abstract the composition specification through the corresponding model. Thereby, we handle the plurality and diversity of the composition approaches.

7 Conclusion and future work

In this paper, we proposed a generic approach to automatically build traces of models composition. We consider the traceability management as a cross-cutting concern. Actually, we designed some graph transformation rules that encapsulate the trace links generation concern. Those rules have been defined around a generic composition metamodel which formalizes the core element of the composition operation. Moreover, a traces generation process, partly independent of the composition language, is introduced to specialize the application of our proposal.

We are currently exploring a pre-configuration support to provide the user with the tool to configure the application of the traceability aspect depending on its purpose. Besides, we intend to work on the traces reusability issue. We have set three possible reuses: validation in model composition, co-evolution of models and optimization of composition chain.

References

1. Laghouaouta, Y., Anwar, A., Nassar, M., Coulette, B.: A graph based approach to trace models composition. JSW **9**(11) (2014) 2813–2822
2. Laghouaouta, Y., Anwar, A., Nassar, M., Bruel, J.M.: On the use of graph transformations for model composition traceability. In: IEEE International Conference on Research Challenges in Information Science (RCIS 2014), IEEE (2014) 1–11
3. Kolovos, D.S., Paige, R.F., Polack, F.A.: Merging models with the epsilon merging language (eml). In: Model Driven Engineering Languages and Systems. Springer (2006) 215–229
4. Jouault, F., Kurtev, I.: Transforming models with atl. In: Satellite Events at the MoDELS 2005 Conference, Springer (2006) 128–138
5. Jouault, F.: Loosely coupled traceability for atl. In: Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability, Nuremberg, Germany. Volume 91., Citeseer (2005)
6. France, R., Ray, I., Georg, G., Ghosh, S.: Aspect-oriented approach to early design modelling. IEE Proceedings-Software **151**(4) (2004) 173–185
7. Ehrig, H., Engels, G., Rozenberg, G.: Handbook of graph grammars and computing by graph transformation. Volume 2. world Scientific (1999)
8. Amar, B., Leblanc, H., Coulette, B.: A traceability engine dedicated to model transformation for software engineering. In: ECMDA Traceability Workshop (ECMDA-TW). (2008) 7–16
9. Grammel, B., Kastenholz, S.: A generic traceability framework for facet-based traceability data extraction in model-driven software development. In: Proceedings of the 6th ECMFA Traceability Workshop, ACM (2010) 7–14
10. Del Fabro, M.D., Bézivin, J., Jouault, F., Breton, E., Gueltas, G.: Amw: a generic model weaver. Proceedings of IDM05 (2005)
11. France, R., Fleurey, F., Reddy, R., Baudry, B., Ghosh, S.: Providing support for model composition in metamodels. In: Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International, IEEE (2007) 253–253
12. Bézivin, J., Bouzitouna, S., Del Fabro, M.D., Gervais, M.P., Jouault, F., Kolovos, D., Kurtev, I., Paige, R.F.: A canonical scheme for model composition. In: Model Driven Architecture–Foundations and Applications, Springer (2006) 346–360
13. Lambers, L., Ehrig, H., Orejas, F.: Conflict detection for graph transformation with negative application conditions. In: Graph Transformations. Springer (2006) 61–76
14. Arendt, T., Biermann, E., Jurack, S., Krause, C., Taentzer, G.: Henshin: advanced concepts and tools for in-place emf model transformations. In: Model Driven Engineering Languages and Systems. Springer (2010) 121–135
15. Vara, J.M., Bollati, V.A., Jiménez, Á., Marcos, E.: Dealing with traceability in the mddof model transformations. IEEE Trans. Software Eng. **40**(6) (2014) 555–583