A Graph Based Approach to Trace Models Composition

Youness Laghouaouta[a], Adil Anwar[b], Mahmoud Nassar[a], Bernard Coulette[c]
[a] IMS-SIME ENSIAS, Mohamed Vth Soussi University, Rabat, Morocco
Email: y.laghouaouta@um5s.net.ma, nassar@ensias.ma
[b] Siweb, EMI, Mohamed Vth Agdal Universtity, Rabat, Morocco
Email: anwar@emi.ac.ma
[c] IRIT-UTM, University of Toulouse II, Toulouse, France
Email: coulette@univ-tlse2.fr

***Abstract*— A model driven engineering process involves different and heterogeneous models that represent various perspectives of the system under development. The model composition operation allows combining those sub-models into an integrated view, but remains a tedious activity. For that, traceability information must be maintained to comprehend the composition effects and better manage the operation itself. Against this context, the current paper describes a framework for model composition traceability. We consider the traces generation concern as a crosscutting concern where the weaving mechanism is performed using graph transformations. A composition specification case study is presented to illustrate our contribution.**

***Index Terms*— traceability, model composition, model transformation, aspect oriented modeling, graph transformation.**

I. Introduction

One of the main Model Driven Engineering (MDE) principles is to reduce system complexity by raising the abstraction level. In MDE, the primary focus is on models rather than computing concepts. Models represent all artifacts handled by the software development process and can be used as first class entities in dedicated model management operations. Therefore, the gap between the requirements definition and the solution is reduced by metamodeling and transformation tools [1].

Usually, complex and large systems are built based on different models; each one representing a view of the system according to a different perspective, a different set of concerns, and a different group of components [2]. The main purpose is to separate concerns in order to represent the software system as a set of less complex sub-models. Hence, the complexity of the analysis/design activities is reduced in the earlier phase of the software development process.

However, several issues are raised, among them the need to synchronize contributing models. This task can be handled through the generation of views that cross different perspectives in order to propagate changes occurring in sub-models. Combining those models can be performed using a model composition approach. Nevertheless, even if model-oriented decomposition is interesting; model composition remains a laborious activity.

Traceability is a necessary system characteristic [3] that reveals the software process maturity. Model composition, as all other model management operations, requires a traceability mechanism for manifold uses: model validation, co-evolution of models and model composition optimization. Indeed, traceability management provides support to better manage the composition operation. It specifies how source artifacts participate in the production of the composed model. Those links detail the flow of execution and are useful to analyze the impact of changing sub-models during the evolution of the system and help to optimize composition chains.

This paper deals with the tracing of the composition of heterogeneous models. Our approach is based on a generic and extensible metamodel accounting for structuring trace links. Essentially, we aim at minimizing the trace links management effort and expressing highly configurable trace models. This paper extends our initial work presented in [4]. It focuses on the generation of traces by using aspect oriented modeling (AOM) principles [5] and graph transformations [6]. In fact, the weaving of the traceability aspect is specified by a set of graph transformation rules.

Graph transformations theory provides a formal support for defining some activities related to model management such as: model transformation, model refactoring and model integration. We intend to populate our traceability metamodel regardless of the composition language. To that end, we believe that graph transformation is a powerful technology for specifying and applying the weaving mechanism of the traces generation code in a composition specification in a more abstract manner.

The rest of the paper is organized as follows: in Section II we review related approaches concerning model transformation traceability; Section III represents an overview of our approach, while Section IV details the generation of the trace model. Thereafter, in Section V we present a concrete working example, followed by a discussion of our contribution in Section VI. Finally Section VII summarizes this paper and presents future works.

doi:10.4304/jsw.9.11.2813-2822

II. Background And Related Work

A. Traceability management in MDE

Traceability is recognized as an essential issue in software engineering, and model driven engineering is no exception. In the literature there are several definitions of traceability, which, differ depending on the artifacts abstraction level and the traceability intensions. The IEEE Standard Glossary of Software Engineering Terminology [7] defines traceability as: *the degree to which relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one other; for example, the degree to which the requirements and design of a given software component match.*

More definitions concerning model transformation traceability have been proposed, notably:

- Dirvalos et al. [3] consider traceability as: *Any relationship that exists between artifacts involved in the software engineering life cycle*.
- Grammel and Voigt [8] define traceability as: *The runtime footprint of model transformation .Essentially, trace links provide this kind of information by associating source and target model element with respect of the execution of a certain model transformation*.

Traceability refers to the ability to capture and reuse links between a set of artifacts handled by a model driven development operation. This information represents the changes that have occurred in these elements and reveals the complexity of logical relations [9] existing among them. In MDE, traceability is a matter of three concerns [10]:

- What: Decide which concepts described in the models will be traced.
- How: Determine how to generate, represent and manage trace links.
- Why: Identify the intentions of capturing trace links.

B. Related work

Several researches address model transformation traceability issue. In this section we briefly outline the main approaches.

Jouault [11] presents an approach to trace transformations written in the ATL language. It addresses the problem of implicit traceability persistence. This approach is the basis for several future researches addressing traceability management. The author considers traces as a model generated in the same way as other target models. The traces generation code can be automatically inserted into any existing ATL program, through the application of a higher order transformation called *traceAdder* [11]. Since the author uses a simple metamodel to represent the trace model structure, traces are not configurable. Nevertheless, the use of the *traceAdder* transformation enhances scalability and allows reusability of the trace model stored externally.

Falleri et al. [12] suggest a framework for traceability of imperative model transformations written in the Kermeta language. The authors consider the trace model as a bipartite graph where the nodes are of two types: source nodes and target nodes. Trace links are stored in a separate model conforms to a generic traceability metamodel and can be reused. Besides, the manual adding of the traces generation code allows the user to select elements to trace, but this reduces scalability.

Amar et al. [13] propose a traceability framework for imperative transformations. The authors present a generic traceability metamodel based on the *"composite"* design pattern, while the trace generation is based on aspect-oriented programming using AspectJ. Thus, it builds traces without modifying the transformation code and supports scalability and reusability of aspects too. The framework defines categories of traceable operations and their respective poincuts. Since it does not take into account all the operations to trace, the programmer can define new custom categories or restrict the predefined ones. Furthermore, the application of the *"composite"* design pattern as well as the link type concept, allow defining configurable trace models.

Grammel and Kastenholz [14] have defined a generic traceability framework for model transformation approaches. It is based on a generic metamodel extensible through facets to simplify hierarchical structure. The approach offers two mechanisms for traces generation: transformation of the implicit trace model to another model conforms to the suggested metamodel and generation of traceability data based on aspect oriented programming. These two mechanisms make the approach scalable and enhance reusability. As for the trace model configuration, it entails the choice of artifacts to trace and the granularity level through the use of facets.

The confusion between model transformation and model composition is a debate topic. Some researches perceive model composition as a transformation with two input models and one output model, when others discern model transformation as a specific model composition which computes the source model with an empty model to produce the target one. Therefore, the presented approaches can be used to trace the model composition operation; however, we judge that the proposal for a model composition traceability approach proves advantageous. Actually, model composition has specific intensions (model synchronization, model integration…) and a particular process (matching step, merging step…) that have to drive the traceability approach.

C. Traceability requirements

In [4], we have detailed an evaluation of the presented approaches based on three comparison criteria: configuration, portability, and scalability. These criteria are inspired from the traceability challenges stated by the Center of Excellence for Software Traceability [15]. According to the results of this analysis, we derived four traceability requirements that have driven our approach.

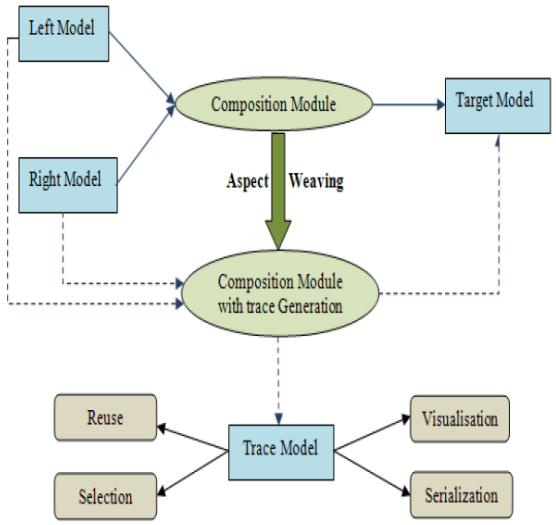

Figure 1. The trace generation process

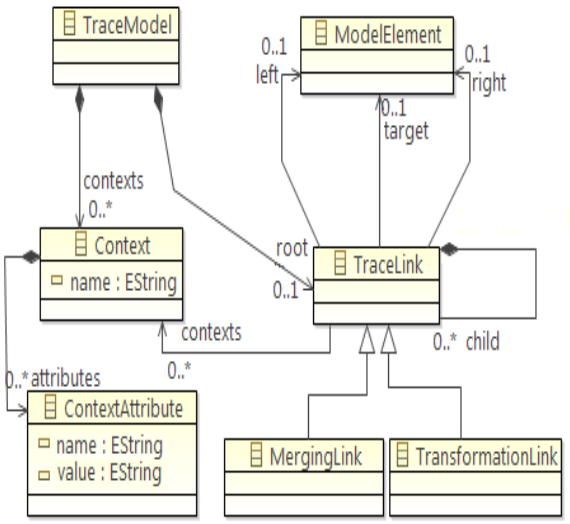

Figure 2. Composition traceability metamodel

- In order to address the scalability challenge, the trace model generation must be automatic; so as to reduce the effort required to achieve traceability. Furthermore, human intervention is useful to configure model elements to trace.
- The code necessary to generate traces must not be intrusive in the primary transformation in order to allow its reuse.
- Traceability data has to be stored in a separate model which conforms to a generic metamodel to reduce trace links management effort. Thus, it supports reusability of the trace model.
- The traceability metamodel has to be expanded with an extensibility mechanism. Essentially, this mechanism allows expressing configurable trace links depending on the traceability scenario and the models specifications.

In our approach, we propose to achieve the two first requirements by using graph transformation rules to generate the trace model. The other points are achieved by using generic composition traceability metamodel to represent the traceability data structure.

III. Overview Of The Approach

A. Trace generation process

We consider the trace model as an additional target model of the composition operation (Fig. 1). To generate it, we propose to use a weaving mechanism of the code responsible of creating traceability elements in the composition specification. We describe in section IV the aspect weaving process in more details. The trace model can be visualized as a graph, or invoked by a selection request. Furthermore, it can be used to validate the composition by checking the consistency and the completeness of the composed model. The co-evolution of models [16] can be supported by analyzing the impact of changing source elements through their corresponding trace links. Finally, we aim to optimize model composition chains; indeed, some trace links are valuable for following steps.

B. The composition traceability metamodel

Several approaches address the model composition operation: AMW [17], EML [18], Kompose [19]. We take into account the typical composition process which involves two major steps: matching and merging. During the first step, similarities between left and right model elements are calculated. Matching elements are merged while other elements are eventually transformed to target model elements or temporary modified to be merged.

We propose a traceability metamodel (see Fig. 2), which defines the different kinds of relationships between model elements independently from any given application domain. We have extended the core traceability metamodel proposed in [4] to support composition traceability requirements and complement it with well-formedness rules. Hereafter, we present the key elements of this metamodel.

A *MergingLink* element connects the left and right elements to the composed element, while a *transformationLink* element represents a transition from each left or right element to the target one. A transformation link may have no source or target element to allow tracing a deleted element or a newly created one. Moreover, we represent multi-scaled trace model that show the imbrications of the rule calls, through a parent-child relation among trace links. The nesting of traces allows the final user to configure the granularity degree he desires.

In order to enhance the trace model semantic richness, we use the *Context* concept to assign additional information to trace links depending on the traceability point of view and the models to compose specifications. Indeed, this extensibility mechanism is based on the definition of the relevant context attributes that capture the further expressiveness data to be assigned to a sub-set of

traces, such as: the composition rule name, the traceability intention...

We present thereafter some well-formedness rules specifying the static semantics of the traceability metamodel.

1) A merging link must have two source elements and one target element.

```
context TraceLink
inv : self.oclIsTypeOf(MergingLink)
implies self.left->notEmpty() and self
    .right->notEmpty() and self.target
    ->notEmpty()
```

2) A transformation link has at most one source element.

```
context TraceLink
inv : self.oclIsTypeOf(
    TransformationLink)
implies  self.left.oclIsUndefined() or
     self.left.oclIsUndefined()
```

3) There is one and only one root trace link of type *MergingLink*.

```
context TraceModel
inv : let root:TraceLink = self.root
    in
if root.oclIsUndefined()
then true
else
root.oclIsTypeOf(MergingLink) and root
    .parent.ocIsUndefined()
endif
```

We illustrate a simple trace model in Fig. 3. The purpose of the composition to trace is to merge two simple class diagrams (*Left* model and *Right* model) each one containing one class *A*. The trace model contains one root element of type *MergingLink* that links the source class diagrams with the target one. Childs of this element represent the merging of the classes *A* and the types *int* in the source models. Finally the copy of the class attributes to the target model is represented by two nested transformation links.

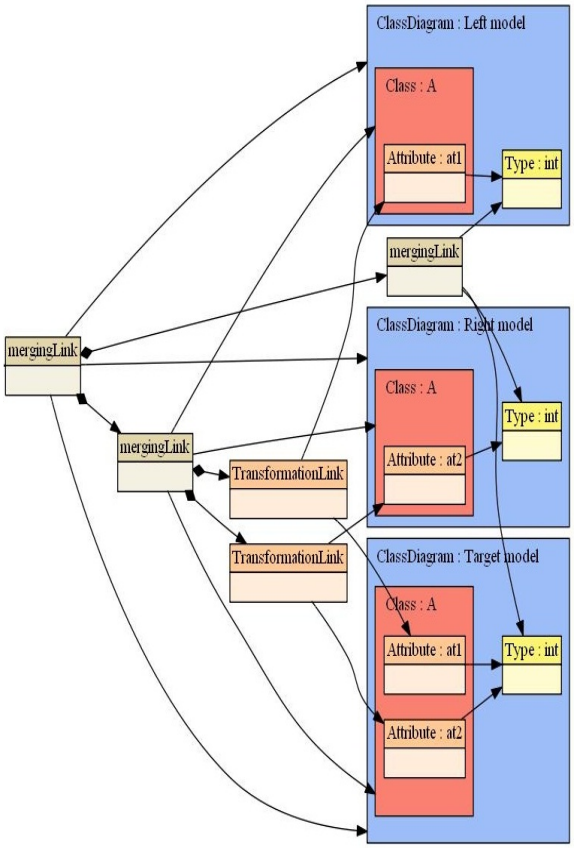

Figure 3. A simple trace model

IV. Trace Model Generation

In this section, we describe how we can use aspect oriented modeling with graph transformations to trace the model composition operation. Our objective is to address our traceability requirements in order to automatically build the trace model without modifying the code of the composition specification by hand. Indeed, we consider the insertion of the trace generation code as a weaving of the base model (which represents the composition specification) with the aspect model (describes the traceability concern). This weaving scenario is specified by graph transformation rules.

A. AOM and graph transformations concepts

Aspect oriented modeling applies aspect oriented programming [20] in the context of MDE, and focuses on modularizing and composing crosscutting concerns during the design phase of a software system. Indeed, the aspect that encapsulates the crosscutting structure and the base model it crosscuts are both models. An aspect is defined principally by:

- A pointcut: it is a predicate over a model used to determine the places where the aspect should be applied (joinpoints).
- An advice: It is the new structure that replaces the relevant jointpoints.

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 left-hand side with the objects of the right-hand side, only if the pattern of the left-hand side can be matched to a given graph [21].

A formal definition of a graph transformation rule is given in [22]: A graph transformation is a rule $r : L \rightarrow R$ from a left-hand side (LHS) graph L to a right-hand side (RHS) graph R. The process of applying r to a graph G involves finding a graph morphism, h, from L to G and replacing $h(L)$ in G with $h(R)$. To avoid dangling edges i.e., edges with a missing source or target node $h(R)$ must be pasted into G in such a way that all edges connected to a removed node in $h(L)$ are reconnected to a replacement node in $h(R)$.

We establish the following correspondences to simulate aspect weaving operation with graph transformation rules: A set of rules correspond to an aspect, the LHS part defines the points where the aspect should be applied

(the pointcut), and the RHS part defines the crosscutting structure that should be inserted at those points (the advice). Note that we have chosen the Henshin project [23] to implement the weaving process.

Henshin is a transformation language and tool environment based on graph transformation concepts and operating on EMF models [23]. It provides features needed to express complex transformation such as: negative application conditions (NACs) which specify the non-existence of model patterns in certain contexts and transformation units to control the rules application sequence.

B. The weaving operation

We have chosen the Epsilon Merging Language EML [18] as an example of dedicated composition language, which is used to express a model merging specification. EML belongs to the Epsilon platform, which is a model driven framework for developing integrated languages for model management tasks such as comparison, transformation, validation, etc. This language proposes to merge models through three categories of rules: match rules, merge rules and transformation rules.

An EML specification can be represented as a graph, since the abstract syntax of EML can be considered as a graph. Hence, the transformation of an EML module to another EML module, which contains traceability generation code, can be considered as a graph transformation. Fig. 4 depicts an excerpt of the EML abstract syntax [24]. Note that the definition of some model elements has been modified to simplify the specification of graph transformations that deal with the generation task.

1) Trace link declaration for merge rules: The rule presented in Fig. 5 allows declaring the traceability element that captures the correspondence between the two source elements matched by the application of an EML merge rule and the merged one. This rule searches for a *MergeRule* node with its connected parameters corresponding to the left, right and target parameters. Thereafter, it adds a new *ParameterDeclaration* node stereotyped with *create*, referencing the merging link to be generated. Besides, the added *AssignStatement* nodes attribute the reference of the corresponding element to the appropriate trace link property (*left*, *right*, and *target*).

2) Trace link declaration for transformation rules: The graph transformation rule presented in Fig. 6 aims to add the trace link declaration to EML transformation rules. As with merge rules, it searches for a *TransformationRule* node and appends to it a new parameter of type *TransformationLink*. This newly added parameter allows generating a trace link that captures the transition from the source element to the target one. Furthermore, the added assign statements attribute the references of the matched *ParameterDeclaration* nodes stereotyped with *preserve* to the generated trace link.

Note that in the EML abstract syntax, no distinction is made between the left and the right elements (the transformation rule connects the source element to the target one). Consequently, we can't automatically resolve

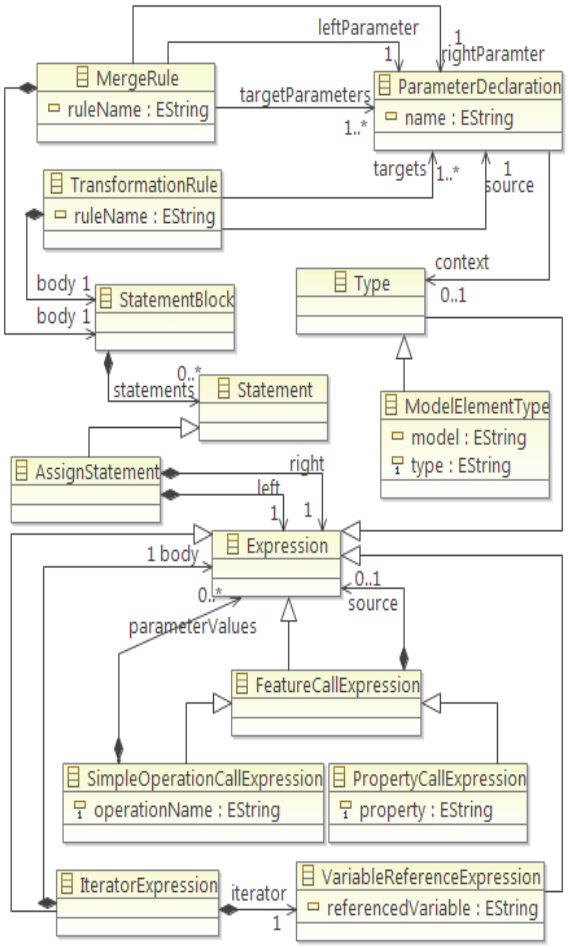

Figure 4. Excerpt of the EML abstract syntax

the origin of the element (left or right model) without user's assistance.

3) Trace links nesting rule: Within EML, the rule call is implicitly performed using the *equivalent* operation that automatically resolves source elements to their transformed counterparts in the target models [24]. This target equivalent is produced by an anterior application of a given rule. We propose to structure traces conforming to the rule invocation sequence. Indeed, the application of the two previous rules allows generating extra-outputs corresponding to trace links that are resolved as potential target equivalents. Hence, we trace a rule call by assigning the trace link generated by the called rule as a child of the link generated by the calling rule.

Accordingly, the rule depicted in Fig. 7 searches for a call of the *equivalent* operation. Thereafter, it copies the reference of the element to resolve (which corresponds to the source of the *SimpleOperationCallExpression* node stereotyped with *delete*) to the variable named *element*. Then, the target equivalents are divided on two subsets: those corresponding to the traceability data that are used to bind the traceability element to its parent and the other element used to copy the original call of the *equivalent* operation. This filtering mechanism is made by applying the *select* operation.

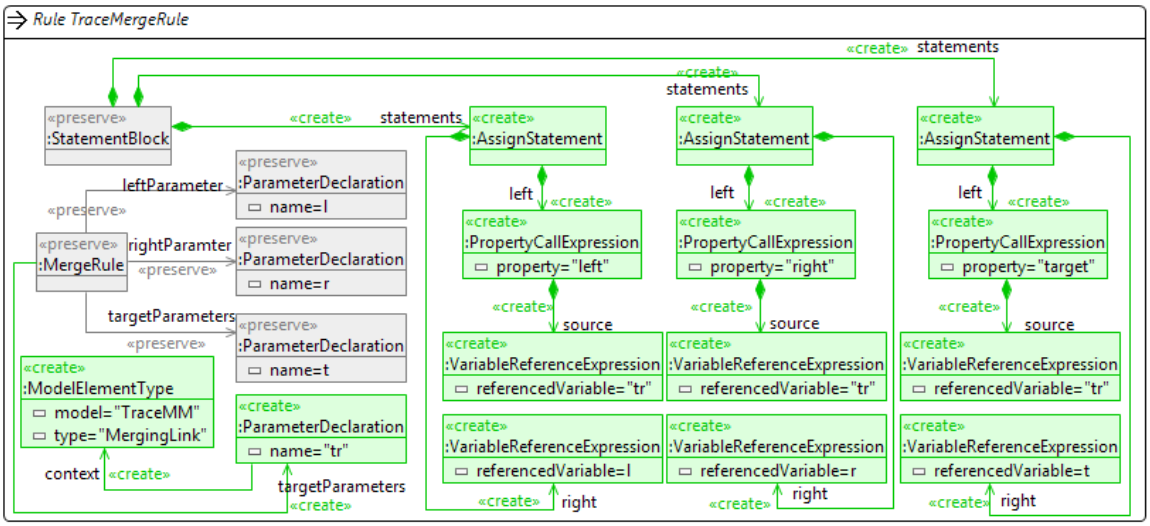

Figure 5. Trace link declaration for merge rules

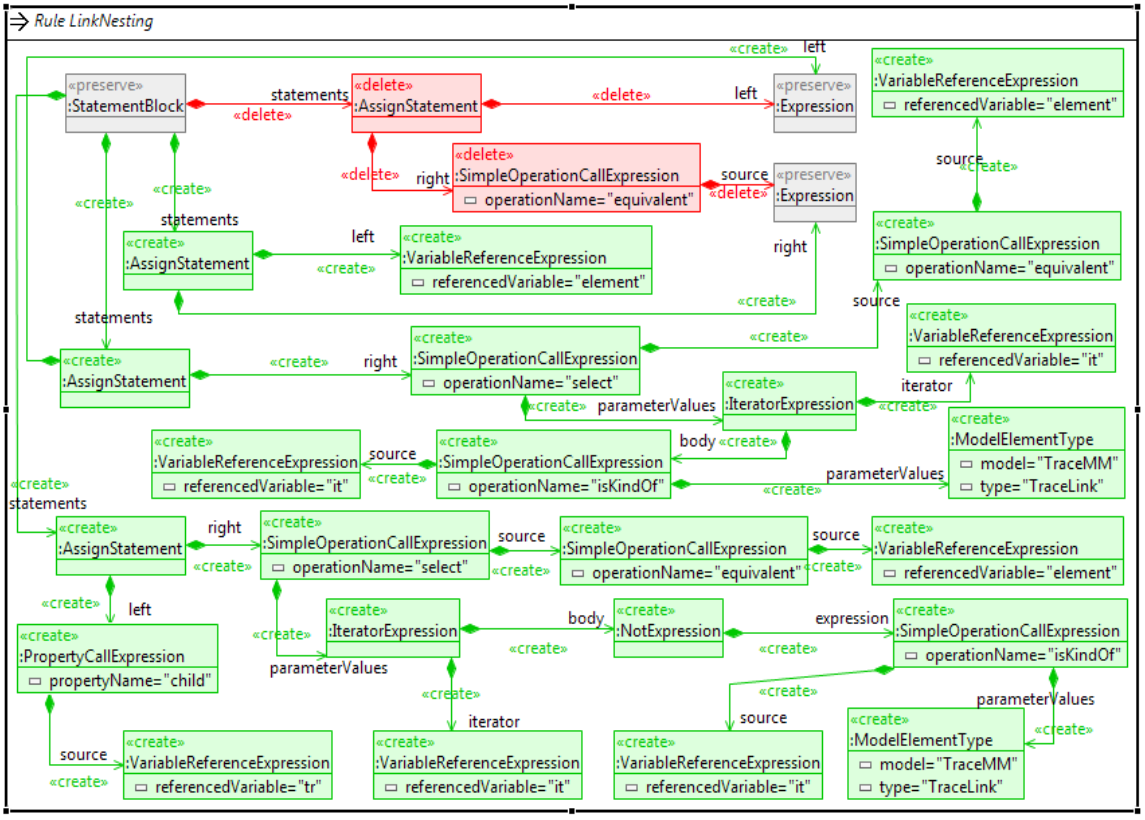

Figure 7. Trace links nesting rule

C. The tool architecture

Fig. 8 depicts a high level view of the tool architecture. Basically, it contains two major layers: the composition and traceability layer and the serialization and visualization layer. The first layer constitutes the core of our architecture while the second one offers facilities to perform the traceability management.

The serialization service is implemented using the EMFText project [25]; it involves a text to model parser and a model to text printer for the EML language. Essentially, this allows transforming the textual EML specification to the corresponding model conforms to the EML abstract syntax. Thereafter, a specific graph transformation unit (which is specified using the Henshin project cf. Section IV-B) weaves the traces generation patterns in the corresponding model. Finally, we reproduce the concrete specification by using the model to text printer.

The execution of the resulting specification generates exta-outputs corresponding to the traceability elements

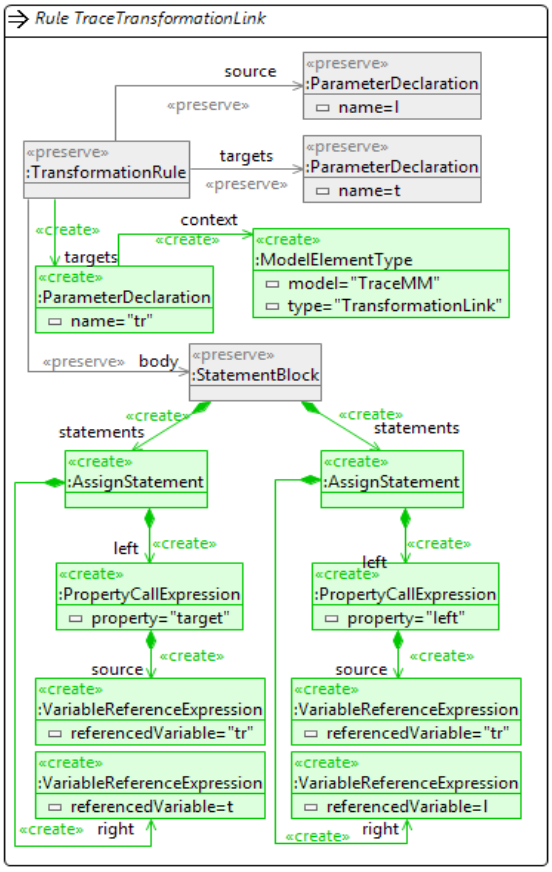

Figure 6. Trace link declaration for transformation rules

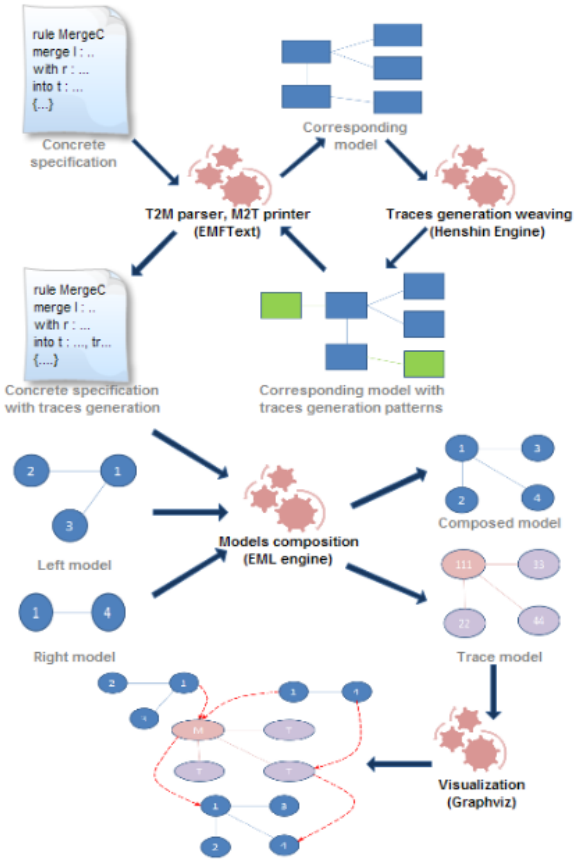

Figure 8. The tool architecture

while producing the composed model. Besides, the visualization service provides support to transform the generated trace model in a human friendly representation.

V. Case Study

In this section, we provide an example to illustrate the application of our approach. The merging scenario we have chosen is the merging of two UML models represented by class diagrams into a target model. The source models as well as the merged model are displayed in Fig. 9. Listing 1 represents the EML rule that merges two source classes, while Listing 2 depicts the resulting modifications over this rule.

```
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 }
```

Listing 1. Merge two classes rule

Depending on the rule type (*merge* or *transformation*), the two first rules of the traces generation weaving unit, declare the traceability parameter as another target parameter, and assign the traceability information to it (Listing

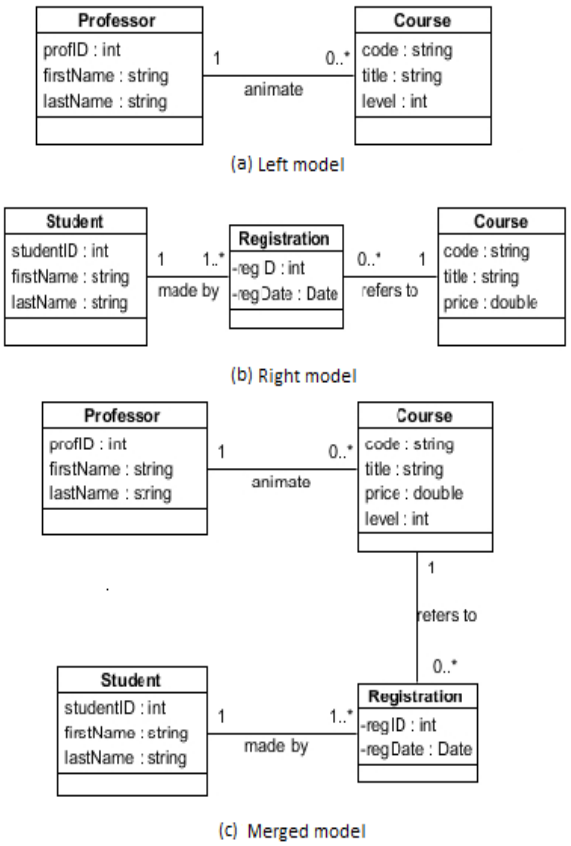

Figure 9. Illustrative example

2: lines 8,11-13). Besides, the call of the *equivalent* operation (Listing 1: line 7) has been captured and replaced with the fragment that divides its return to trace model elements and default target elements (Listing 2: lines 1-4,14-16). The first sub-set is used to copy the original call of the *equivalent* operation (Listing 2: line 15), while the traceability element is assigned as a child of current trace link (Listing 2: line 16).

```
1  pre
2  {
3  var element : new Any ;
4  }
5  rule MergeClassWithClass
6  merge l : left!Class
7  with r : right!Class
8  into t : target!Class , tr:trace!MergingLink
9  {
10 t.name = l.name;
11 tr.left=l;
12 tr.right=r;
13 tr.target=t;
14 element = l.ownedAttribute.includingAll(r.
       ownedAttribute);
15 t.ownedAttribute = element.equivalent().select(
       it | not it.isKindOf(trace!TraceLink));
16 tr.child = element.equivalent().select(it | it.
       isKindOf(trace!TraceLink));
17 }
```

Listing 2. Merge two classes with traces generation

Fig. 10 depicts an excerpt of the generated trace model. This model conforms to our composition traceability metamodel and contains two types of trace links (merging links and transformation links) that are generated with respect to the composition relationships kinds. Those links are nested with respect to the rules invocation sequence. Essentially, the multi-scaled character of trace links allows the user to navigate over the trace model, from rough to precise. Note that we have used the Emf2gv project[1] to visualize the trace model.

VI. Discussion

As presented in section II, traceability involves three concerns: what to trace, how to manage the traceability information, and why we require it. Our approach allows the user to identify a subset of elements to trace through the selection of the relevant aspects (graph transformation rules) to apply. On the other hand, the use of aspect oriented modeling and graph transformation rules automatically insert the code responsible for generating the traceability information. In order to reduce effort to achieve traceability and support reusability of aspects, the trace model conforms to a generic metamodel. Besides, our metamodel is extensible to express traces regarding traceability scenarios. Finally, we identified three major intensions of capturing traces: validation in model composition, co-evolution of models and optimization of composition chains. These intensions will guide our future work. We consider that the challenge is to make our approach aware of the "why", in order to automatically select the elements to trace and configure the trace management process depending on the user's intension.

The use of graph transformation proves to be advantageous for augmenting composition tools with a traceability support, since, existing graph based tools as Henshin project can perform this operation. It provides features needed to express complex transformation such as: application conditions and the control flow of graph transformation rules. Furthermore, the plurality of the composition languages and their characteristics (textual, model-based, and graph-based) make traceability difficult to manage; however, we believe that the exploration of graph transformation options provides ways to overcome this problem.

We aim to abstract as much as possible the composition specification to the corresponding graph. Thereby, our approach can be used to trace model composition regardless its nature: textual specifications written in EML, model-based specification in ATL, and graph based composition [2]. However, our contribution is currently a language dependent approach, since the definition of the graph transformation rules takes into account the composition language. As a solution, we are considering a pivot language.

VII. Conclusion And Future Work

In this paper, we presented an approach dedicated to manage the traceability concern in a model composition operation. Our solution fits a set of traceability requirements we have deduced from the analysis of the main model transformation traceability approaches. Indeed, we consider traceability as a cross-cutting concern and we generate the trace model automatically based on the aspect oriented paradigm. The aspect weaving is implemented using graph transformation rules. Moreover, we use a generic and extensible traceability metamodel that deals with the configuration challenge.

Several perspectives to our work are under consideration. We are completing the visualization of the generated trace model in a human-friendly representation using the graphviz tool [26]. Besides, we intend to work on a pre-configuration tool support to generate the trace model according to the user's requirements. Finally, we believe that traceability data is useful for optimizing the establishment of correspondences between contributing models, by automatically refining the matching model.

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

References

[1] S. Kent, "Model driven engineering," in *Integrated formal methods*. Springer, 2002, pp. 286–298.

[2] A. Anwar, A. Benelallam, M. Nassar, and B. Coulette, "A graphical specification of model composition with triple graph grammars," in *Model-Based Methodologies for Pervasive and Embedded Software*. Springer, 2013, pp. 1–18.

[3] N. Drivalos, R. F. Paige, K. J. Fernandes, and D. S. Kolovos, "Towards rigorously defined model-to-model traceability," in *ECMDA Traceability Workshop (ECMDA-TW'08)*, 2008, pp. 17–26.

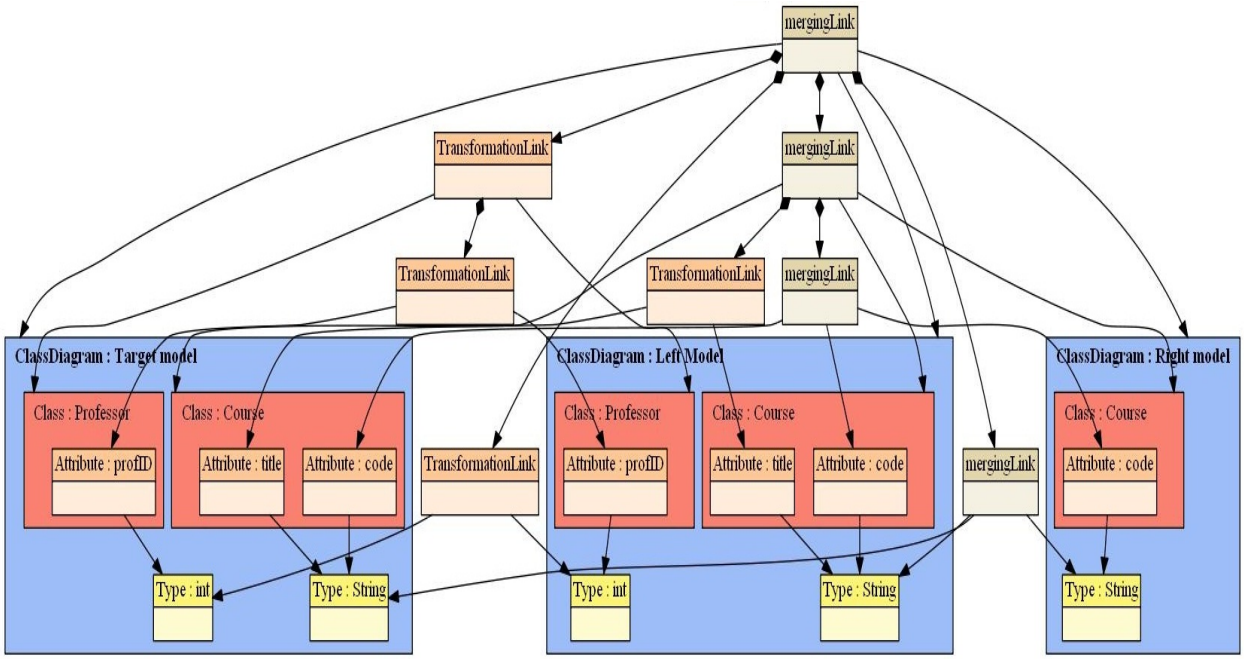

Figure 10. Excerpt of generated trace model

[4] Y. Laghouaouta, A. Anwar, and M. Nassar, "A traceability approach for model composition," in *2013 ACS International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 2013, pp. 1–4.

[5] R. France, I. Ray, G. Georg, and S. Ghosh, "Aspect-oriented approach to early design modelling," *IEE Proceedings-Software*, vol. 151, no. 4, pp. 173–185, 2004.

[6] G. Rozenberg and H. Ehrig, *Handbook of graph grammars and computing by graph transformation*. World Scientific Singapore, 1997, vol. 1.

[7] J. Radatz, A. Geraci, and F. Katki, "Ieee standard glossary of software engineering terminology," *IEEE Std*, vol. 610121990, p. 121990, 1990.

[8] B. Grammel and K. Voigt, "Foundations for a generic traceability framework in model-driven software engineering," in *ECMDA Traceability Workshop (ECMDA-TW'09)*, 2009.

[9] N. Anquetil, B. Grammel, I. Galvão, J. Noppen, S. S. Khan, H. Arboleda, A. Rashid, *et al.*, "Traceability for model driven, software product line engineering," in *ECMDA Traceability Workshop (ECMDA-TW'08)*, 2008, pp. 77–86.

[10] G. Spanoudakis and A. Zisman, "Software traceability: a roadmap," *Handbook of Software Engineering and Knowledge Engineering*, vol. 3, pp. 395–428, 2005.

[11] F. Jouault, "Loosely coupled traceability for atl," in *ECMDA Traceability Workshop (ECMDA-TW'05)*, vol. 91. Citeseer, 2005.

[12] J.-R. Falleri, M. Huchard, C. Nebut, *et al.*, "Towards a traceability framework for model transformations in kermeta," in *ECMDA Traceability Workshop (ECMDA-TW'08*, 2006, pp. 31–40.

[13] B. Amar, H. Leblanc, and B. Coulette, "A traceability engine dedicated to model transformation for software engineering," in *ECMDA Traceability Workshop (ECMDA-TW'08)*, 2008, pp. 7–16.

[14] B. Grammel and S. Kastenholz, "A generic traceability framework for facet-based traceability data extraction in model-driven software development," in *ECMDA Traceability Workshop (ECMDA-TW'10)*, 2010, pp. 7–14.

[15] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, and J. Maletic, "The grand challenge of traceability (v1. 0)," in *Software and Systems Traceability*. Springer, 2012, pp. 343–409.

[16] B. Amar, H. Le Blanc, P. Dhaussy, B. Coulette, *et al.*, "Trace transformation reuse to guide co-evolution of models," in *5th Int. Conference on Software and Data Technologies (ICSOFT'10)*, 2010.

[17] M. D. Del Fabro, J. Bézivin, F. Jouault, E. Breton, G. Gueltas, *et al.*, "Amw: a generic model weaver," *Procs. of IDM05*, 2005.

[18] D. S. Kolovos, R. F. Paige, and F. A. Polack, "Merging models with the epsilon merging language (eml)," in *Model Driven Engineering Languages and Systems*. Springer, 2006, pp. 215–229.

[19] F. Fleurey, B. Baudry, R. France, and S. Ghosh, "A generic approach for automatic model composition," in *Models in Software Engineering: Workshops and Symposia at MODELS 2007*, vol. 5002. Springer, 2008, p. 7.

[20] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *ECOOP*, 1997, pp. 220–242.

[21] L. Lambers, H. Ehrig, and F. Orejas, "Conflict detection for graph transformation with negative application conditions," in *Graph Transformations*. Springer, 2006, pp. 61–76.

[22] J. Whittle, J. Araújo, and A. Moreira, "Composing aspect models with graph transformations," in *Proceedings of the 2006 international workshop on Early aspects at ICSE*. ACM, 2006, pp. 59–65.

[23] T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer, "Henshin: advanced concepts and tools for in-place emf model transformations," in *Model Driven Engineering Languages and Systems*. Springer, 2010, pp. 121–135.

[24] D. Kolovos, L. Rose, R. Paige, and A. Garcıa-Domınguez, "The epsilon book," *Structure*, vol. 178, 2010.

[25] F. Heidenreich, J. Johannes, S. Karol, M. Seifert, and C. Wende, "Derivation and refinement of textual syntax for models," in *Model Driven Architecture-Foundations and*

Applications. Springer, 2009, pp. 114–129.
[26] E. R. Gansner, "Drawing graphs with graphviz," Technical report, AT&T Bell Laboratories, Murray, Tech. Rep., 2009.

Youness Laghouaouta received the Engineer of state degree in Software Engineering from National High School of Computer Science and Systems Analysis (ENSIAS) in 2009. He is currently a PhD student in the IMS (Models and Systems Engineering) Team of SIME Laboratory at ENSIAS. His research interests are model traceability, model composition, Aspect Oriented Engineering, and Model-Driven Engineering.

Adil Anwar works as an assistant professor in computer science at the university of Mohammed-V Rabat, and as a member of the Siweb research team of Mohammadia school of engineers. In 2009, he received a Ph.D degree in Computer Science at the University of Toulouse. He is interested in software engineering, including model driven software engineering, mainly by heterogeneous software language modelling, traceability management in MDE, combining formal and semi-formals methods in software development.

Mahmoud Nassar is Professor and Head of the Software Engineering Department at National Higher School for Computer Science and Systems Analysis (ENSIAS), Rabat, Morocco. He is also Head of IMS (Models and Systems Engineering) Team of SIME Laboratory. He received his PhD in Computer Science from the INPT Institute of Toulouse, France. His research interests are integration of viewpoints in Object-Oriented Analysis/Design (VUML profile), Model-Driven Engineering, and Context-Aware Service-Oriented Computing.

Bernard Coulette works as a full professor at the University of Toulouse, and as a member of the MACAO team of IRIT laboratory. His research fields of interest are mainly integration of viewpoints in Object-Oriented Analysis/Design (VUML profile), modeling and enactment of Model Driven Processes. He has directed several PHD thesis in the context of international collaborations (Vietnam, Morocco).