A dedicated approach for model composition traceability

Youness Laghouaouta, Adil Anwar, Mahmoud Nassar, Bernard Coulette

A dedicated approach for model composition traceability

Youness Laghouaouta[a,*], Adil Anwar[b], Mahmoud Nassar[a], Bernard Coulette[c]

[a] IMS Team, ADMIR Laboratory, Rabat IT Center, ENSIAS, Mohammed V University in Rabat, Morocco
[b] SIWEB Team, EMI, Mohammed V University in Rabat, Morocco
[c] IRIT Laboratory, University of Toulouse, France

Keywords:
Model traceability
Model composition
Aspect-oriented modeling
Graph transformations
NFR framework

A B S T R A C T

Context: Software systems are often too complex to be expressed by a single model. Recognizing this, the Model Driven Engineering (MDE) proposes multi-modeling approaches to allow developers to describe a system from different perspectives. In this context, model composition has become important since the combination of those partial representations is inevitable. Nevertheless, no approach has been defined for keeping track of the composition effects, and this operation has been overshadowed by model transformations.

Objective: This paper presents a traceability approach dedicated to the composition of models. Two aspects of quality are considered: producing relevant traces; and dealing with scalability.

Method: The composition of softgoal trees has been selected to motivate the need for tracing the composition of models and to illustrate our approach. The base principle is to augment the specification of the composition with the behavior needed to generate the expected composed model accompanied with a trace model. This latter includes traces of the execution details. For that, traceability is considered as a crosscutting concern and encapsulated in an aspect. As part of the proposal, an Eclipse plug-in has been implemented as a tool support. Besides, a comparative experiment has been conducted to assess the traces relevance. We also used the regression method to validate the scalability of the tool support.

Results: Our experiments show that the proposed approach allows generating relevant traces. In addition, the obtained results reveal that tracing a growing number of elements causes an acceptable increase of response time.

Conclusion: This paper presents a traceability approach dedicated to the composition of models and its application to softgoal trees. The experiment results reveal that our proposal considers the composition specificities for producing valuable traceability information while supporting scalability.

1. Introduction

Traceability consists in linking the software development artifacts to each other through specific relationships [1]. Its practice is recognized as an essential activity that enhances quality aspects of the final solution (e.g. efficiency, maintainability, ...). For example, functional coverage analysis can be achieved by exploring traceability relationships between requirements and their realizations [2]. Also, relations between design products and their implementations provide a support for optimizing maintenance tasks and analyzing impacts of changes [3–5].

Moreover, the complexity of current systems makes it essential to use a traceability support. The number of software artifacts is increasing significantly, traceability management helps mastering this complexity by exposing intrinsic relationships and by recording the life cycle of each artifact. However, this legitimate need for traceability practice and the various advantages it offers do not confer a wide scale use. This is due, essentially, to the additional cost of capturing and maintaining traceability links, and the divergence of interests between the links creators and users [6].

The requirements engineering community was the first to invest in traceability integration. Its purpose being to ensure that the requirements are well covered by the final product, traceability is presented as a support to such assessment. Thereafter, the traceability benefits have attracted the interest of Model Driven Engineering (MDE) researches. In this software engineering field, other aspects of traceability management have emerged. Those aspects depend in terms of traceability intentions (the users' expectations by its practice) and the nature of the elements to be traced (model

* Corresponding author.
E-mail address: y.laghouaouta@um5s.net.ma (Y. Laghouaouta).

http://dx.doi.org/10.1016/j.infsof.2017.07.002

or non model artifacts). Essentially, two classes of traceability approaches were identified [7]: requirements traceability and model transformations traceability.

Indeed, in a Model Driven Development (MDD) process, the final system is obtained by transforming requirements towards solutions that realize them. These solutions are not directly produced through one shot operation, but they result from sequential transformations applied to the primary models. Accordingly, requirements traceability specifically addresses the management of relationships between requirements and the corresponding solutions. While model transformations traceability focuses on capturing the effects of executing model transformations against the managed models. Nevertheless, the model transformation operation does not constitute the only pillar of MDD. The composition operation is also of major importance as it supports multi-modeling approaches. Indeed, to reduce the complexity of the development activity, developers can describe the system by as many models as they want. These partial representations will be less complex than the global model. However, they need to be composed to deal with model validation and synchronization issues, and to better understand the interrelations between them.

Nonetheless, the composition of models remains a complex activity. In order to overcome this issue, traceability practice is presented as a significant solution. Indeed, traces expose the exact effects of executing the composition and help developer to better comprehend interrelations among managed models. Also, they provide a basis for validating the composition (e.g. each element of the composed model has to be connected with a traceability link which justifies its existence) and evolving models when changes occur (e.g. elements that are impacted by removing a given source model element can be retrieved from analyzing traceability links they are connected with). Taking full benefits of the aforementioned possible exploitations of traceability information implies the capture of expressive and interesting traces. However, given that existing transformation traceability supports disregard the composition features; their application to composition scenarios compromises the expressiveness of traces and leads to the production of traceability information almost worthless.

Recognizing this, the paper presents a traceability management approach dedicated to the model composition operation. Our proposal is applied to model compositions specified with a rule based language. The principal is inspired from the work presented by Jouault [8] and it consists in allowing the composition to generate traces like other target elements. Hence, traces that expose the effects of applying the composition specification to the source models are constructed similarly and in parallel to the construction of the composed model elements.

Fig. 1 illustrates this idea. The basic artifact of the traces generation process is a preexisting specification that allows generating the composed model. The principle is to augment this specification with the behavior it needs to generate traceability information. After that, the execution of the resulting specification will produce two outputs: the default composed model, and the corresponding trace model. This latter conforms to a generic traceability metamodel that provides the concepts needed to express the expected traceability information. For implementing such mechanism, we consider traceability as a crosscutting concern that is encapsulated in an aspect. The application of this aspect (by means of a dedicated weaving process) weaves the traces generation patterns into the primary composition specification.

In previous works [9,10], we presented the specialization of this traces generation process for the Epsilon Merging Language EML [11] and the ATLAS Transformation Language ATL [12]. We described the corresponding traces generation patterns as well as mechanisms to perform the weaving of these patterns into the

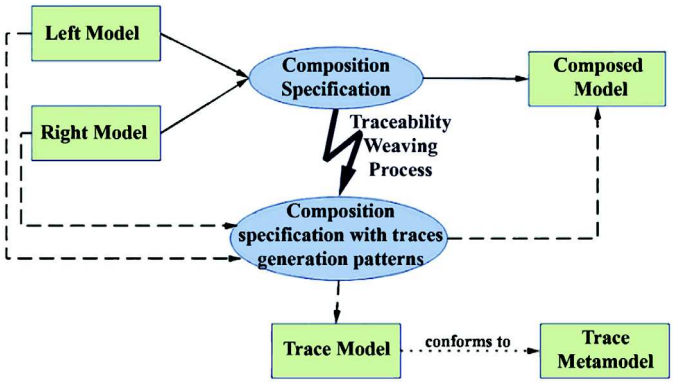

Fig. 1. Global overview of the traces generation process.

composition specifications. The current paper focuses on the validation of the proposed traceability approach. By using the *ComposeTracer* plugin (i.e. the tool support we implemented for the proposed traceability approach) we realized different composition scenarios. After that, we calculated and compared the values taken by some appropriated metrics to validate the relevance of the generated traces. Also, we relied on a statistical method for analysis (i.e. linear regression) to predict the variation of response time depending on the number of elements to be traced. For convenience, we summarize the specific contributions of this paper in three points:

- Composition of softgoal refinement trees which has been selected to demonstrate the need for a traceability approach dedicated to the model composition operation.
- Implementation details of the tool supporting the proposed traceability approach.
- Results of an empirical study that aims to assess the traces relevance and the scalability of the traceability support.

The remainder of this paper is structured as follows. In Section 2, we introduce our traceability approach. Section 3 presents an illustrative example that motivates the need for a dedicated composition traceability approach. Section 4 gives a background to traceability management and lists a set of requirements that have driven our approach. Section 5 provides an overview of our conceptual proposal. In Section 6, we explain implementation details of the *ComposeTracer* plug-in which supports the traces generation. Section 7 presents the validation of the proposal. Section 8 discusses our contributions to overcome some traceability challenges. Section 9 lists the related works. Finally, Section 10 summaries this paper and presents future work.

2. Overview of the proposed traceability approach

The aim of this section is to provide a global overview of our traceability approach dedicated to the composition of models. Basically, such approach for generating traces is applied on the composition of any type of software artifacts models (e.g. requirements models, design models, implementation models). However, given the type of traceability information produced by our approach and especially its granularity level, the proposed approach is mainly intended for analysts and designers. Implementation models are excluded since maintaining their traceability requires fine grained supports.

Recalling from Section 1, the underlying idea is to allow the composition specification to generate and record traces in an extra-output model which accompanies the expected composed model. For that, the composition specification (it consists of a set of

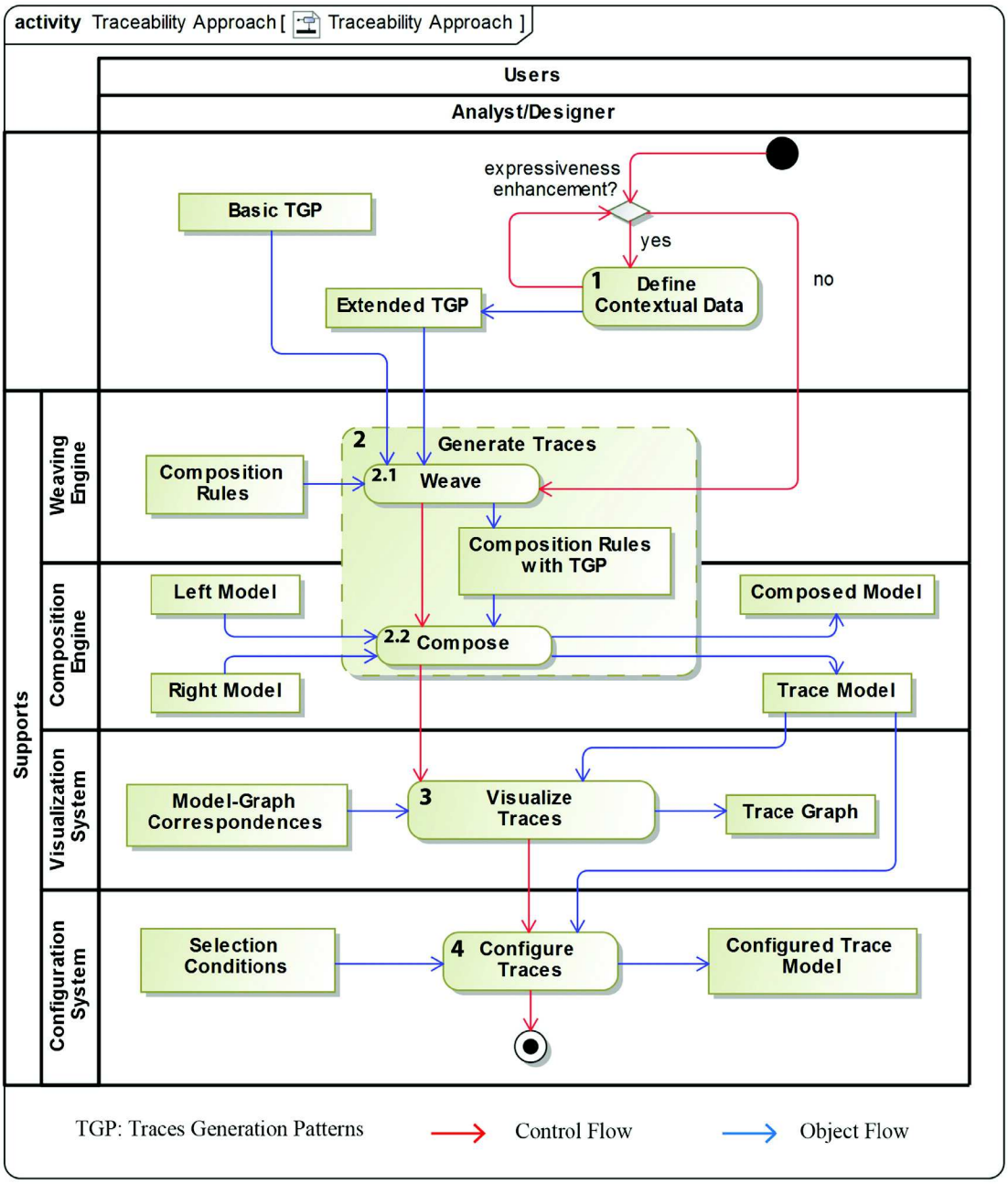

Fig. 2. Global overview of the proposed traceability approach.

composition rules that define relationships between the source and composed elements) has to be augmented with the behavior needed to produce the trace model.

Indeed, for a given composition language, some traces generation patterns have to be defined around its concepts. Essentially, for each specific composition rule type, we define the corresponding structure to be added to the rule instances for allowing them to produce traces upon their applications. This implies an internal maintenance of the tool supporting the proposed approach and the new structure is independent of the involved metamodels (i.e. the metamodels of the left, right and composed models). Hence, such basic traces generation patterns can be added to various specifications expressed with the composition language.

On the other side, we offer the possibility to enhance the expressiveness of the produced traces by defining extended traces generation patterns (see Fig. 2). For a given composition scenario, the analyst/designer can define the expressive data deemed interesting to be captured (**Action** 1). In this case, the related generation patterns may exploit the concepts defined in the involved metamodels (e.g. values taken by an attribute characterizing a meta-class) which restrict the using scope of these patterns to the specific composition scenario.

To allow the analyst/designer generating traces upon the execution of a composition specification (**Action 2**), the basic and extended traces generation patterns are weaved into the primary composition rules (more details about the weaving mechanisms are given in Sections 5.2 and 6.2.2). The application of the resulting composition rules uses the composition engine (e.g. EML engine, ATL engine ...)) and it produces the expected composed model accompanied with the trace model.

In some scenarios, the trace model may comprise a large amount of details which compromises the examination of traces. As a solution, our approach relies on a visualization system (cf. Section 6.2.4) to represent the trace models like graphs. For that, the analyst/designer has to specify correspondences between the abstract elements defined in the involved metamodels and the graphical representations they have to be displayed like (**Action 3**). The second solution is to limit the examination to a sub-set of traces. In this case, the analyst/designer expresses some selection conditions for the interesting traces (**Action 4**).

The ability to produce trace models, to visualize them, and to extract interesting fragments are considered. The remaining paramount activity is the exploitation of traceability information. Indeed, trace models are subject to model management operations to support diverse activities. Among those, we are specifically interested in the model evolution and the verification of consistency.

3. Need for a dedicated model composition traceability approach: a motivating example

The aim of this section is to motivate the need for a traceability approach dedicated to the composition of models. The illustrative example we have chosen is the composition of softgoal trees. These latter allow representing refinements of Non-Functional Requirements (NFRs) under the NFR framework [13]. We selected this scenario because problems of handling separated softgoal trees are clearly identified in the literature [14], and admitting that traceability is a key factor to address them.

We will first provide a brief overview of NFR framework and present the managed models. Thereafter, we will introduce the specification that is used to perform the composition scenario. Then, we will exploit the presented elements to illustrate to what extend it is required to design a traceability approach which considers the composition specificities.

3.1. Illustrative example

3.1.1. The NFR framework

Non-functional requirements describe the customers' expectations of how the system should meet functional requirements [15] (e.g. performance, security, ...). Since the completion or failure of those needs is tied to achieving software quality [15], their satisfaction is very important to be held into account from the earlier stages of the software development process. With a view to anticipating the consideration of such requirements, the NFR Framework [13] provides a framework for thinking about ways to meet those needs. It is a goal-oriented approach that aims to make available to analysts a support for modeling non-functional requirements making explicit their representation. Also an analytical approach is described for graphical treatment of requirement satisfaction. This latter is based on requirement refinement, interdependencies analysis, establishment of priorities, and the description of ways to satisfy NFRs. Indeed, this framework considers the NFRs as goals to be achieved (softgoals) and provides three levels of refinement for these:

- *NFR softgoals*: They represent well identified non-functional requirements and express the first level of softgoals refinement.
- *Operationalizing softgoals*: They describe the ways in which their parents' goals can be met (methods, technical solutions, accepted functionalities, ...).
- *Claim softgoals*: They allow marking priorities between softgoals and introducing arguments to justify a given refinement.

The refinement of softgoals in more fine grained ones as well as contributions that have the realization of certain softgoals on others are represented by a goal tree called SIG (Softgoal Interdependency Graph). This graph comprises a decomposition of softgoals using *AND/OR* relationships. Whereas the satisfaction degree that has the realization of an *operationalizing softgoal* on its softgoal parent is expressed by a so called "*satisficing*" relationship. Four types of contributions are generally used: "*Make*" if a softgoal is completely fulfilled through its children (called *offspring*); "*Help*" when meeting a softgoal positively satisficing another; the type "*Hurt*" is used to describe cases where the realization of an *operationalizing softgoal* compromises its parent; and the type "*Break*" expresses that the contribution is completely negative. All these concepts were defined by Supakkul and Chung in a UML profile [16]. In what follows, we present examples of SIG models that illustrate the use of these concepts.

3.1.2. On the SIGs composition

The NFR framework aims to provide a detailed view on the NFRs' satisfaction. However, several SIGs are generally elaborated, and they can express complementary viewpoints for some softgoals but contradictory for certain contributions. Therefore, the need for composing them in an integrated and more expressive view arises. Wei et al. [14] were interested in this composition scenario to address the limitations of handling separated SIGs under the NFR framework:

- The stakeholder knowledge limitation can cause potential conflicts in softgoals refinements. Therefore, a concentration on a specific viewpoint leads to inconsistency and incompleteness problems.
- The impact analysis of making a decision to consider or discard a given softgoal. For example, a SIG can express a positive contribution that has the realization of a given *operationalizing softgoal*, but in another model, its realization can be marked as preventing the achievement of more critical requirements.

The composition mechanism proposed by Wei et al. [14] separates the source models in a base SIG and an extension SIG. Indeed, the first model is enriched by refinements concerning its softgoals which are expressed in the opposite model. We are interested in this scenario as a basis for the illustration of our traceability approach for two main reasons. On the one hand, both the composition mechanism and the intended purposes are clearly identified. On the other hand, if the SIGs composition offers, certainly, a means of addressing the aforementioned limitations, the recourse to traceability presents a compelling solution for better manage these issues.

The composition scenario operates on two SIGs that we have designed. They express refinements for the "Security" and "Performance" NFRs in the context of database management. The base model presents a decomposition of security which combines the management of data integrity and confidentiality (Fig. 3). Regarding the second point, establishing a control over the use of database is the principal aspect to be considered. This includes the control of resources and access as well as keeping track of the relevant information through logging techniques. As for data integrity, it is satisfied by data validation to ensure data consistency and encryption to affirm its non-alteration or destruction.

The refinement of security (Fig. 3) will be enriched by the knowledge expressed in the model depicted in Fig. 4. Minimizing the response time is one of the requirements to be considered in order to enhance performance. By contrast, the logging technique reduces the system performance as it supposes analyzing transactions. For the same reason, the data validation, spread across the two presented aspects (preserving referential integrity, and implementing customized business checks through triggers), is an *Operationalizing softgoal* to avoid. Finally, indexing is proposed as a solution for minimizing the response time.

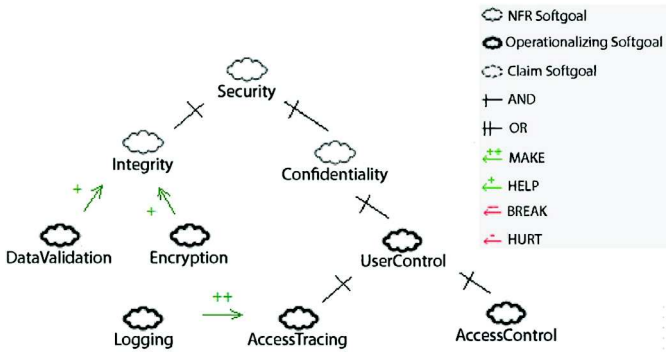

Fig. 3. Refinement of Security under the NFR Framework (base SIG).

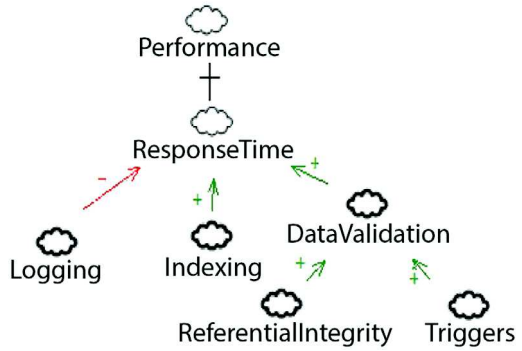

Fig. 4. Refinement of Performance under the NFR Framework (extension SIG).

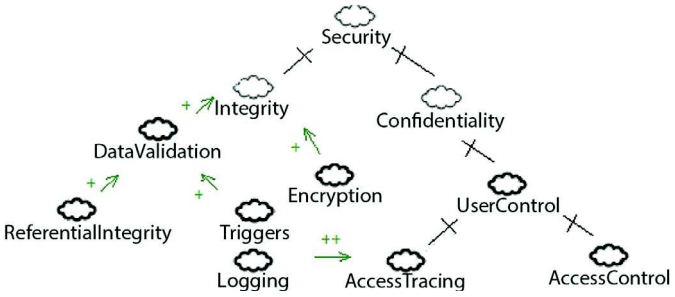

Fig. 5. Composed SIG.

The model resulting from the composition of these two SIGs is depicted in Fig. 5 (the specification used to automatically produce the composed model will be detailed in the next section). Since the composition intention is to enrich the base SIG, all the elements it includes (softgoals and contributions) were transcribed in the composed model. Moreover, we can recognize that the softgoal "DataValidation" is extended with the knowledge available in the second SIG. Indeed, the relevant refinement branch is now attached to its equivalent softgoal in the target model.

It is obvious that the resulting refinement of the "Security" NFR is more expressive in comparison with the base version. However, this model does not express all effects of a decision-making for the realization or the omission of a given softgoal on the whole non-functional requirements. For example, the composed SIG outlines the positive contribution that has the logging technique on database security. However, the softgoals refinement expressed in the extension SIG stipulates that this technique compromises the performance aspect. Against this lack of expressiveness, traceability proves to be interesting. It allows analysts/designers to reason about the effective contributions of softgoals belonging to the composed SIG in their relations with those of partial SIGs. On the other side, traceability will present a detailed view on the composition execution. This information is valuable for validating the composition and optimizing the models synchronization tasks when changes occur.

3.1.3. The composition specification

We specify the composition of SIG models with the dedicated merging language EML [11]. Furthermore, to simply express the composition specification, the handled models will conform to a metamodel defined around the SIG profile [16]. Our metamodel depicted in Fig. 6 expresses the key concepts needed to construct the models presented in Section 3.1.2. Indeed, it focuses on softgoals of types *NFRSoftgoal* and *OperationalizingSoftgoal*. As for the stereotypes *SIG, Proposition, Contribution, SatisficingCT* and *DecompositionCT* which are defined in the SIG profile, we translated them into their equivalent meta-classes. An exception is the *Type* stereotype that we have replaced with an attribute characterizing the *Softgoal* meta-class.

A composition specification in the Epsilon platform [17] is divided into two separate scripts: the comparison and the merging modules. In Appendix A, we provide the script ECL (Epsilon Comparison Language) [17] that allows establishing correspondences between the source SIG models. It operates only on softgoals (instances of the *NFRSoftgoal* and *OperationalzingSG* metaclasses), and does not compare contributions (instances of *DecompositionCT* and *SatisficingCT*). After the comparison step, two categories of elements are identified: softgoals of the base model that have a corresponding in the opposite model, and propositions (softgoals and contributions) with no corresponding.

In order to exploit these correspondences, the merging script (written in EML) combines a set of declarative and imperative rules (cf. Appendix A). The principle is to copy all elements of the base model by following a declarative approach. Besides, each time a pair of softgoals is merged, the operations which connect the extension branch to the constructed softgoal are triggered. Thus, the *TransformNFRSoftgoal* and *TransformOperationalizingSG* rules transform softgoals of the base SIG that have no corresponding in the second one. And given that no comparison rule is applicable for contributions, the *TransformDecompositionCT* and *TransformSatisficingCT* rules transcribe all contributions belonging to the base SIG into the composed one. While the *MergeNFRSoftgoal* and *MergeOperationalizingSG* rules allow merging corresponding softgoals and producing their target equivalents in the composed SIG.

The EML language define an *equivalent*() operation that makes possible to express rule calls. This operation plays a predominant role in expressing the specification that composes the SIG models. It allows linking elements generated in a declarative way with their successors (*offsprings*) produced by a same way (this is considered as an implicit rule call mechanism). However, if the equivalent offspring is not identified (it was not generated by applying a declarative rule), the *equivalent*() operation requests a suitable rule (a lazy rule) to produce this equivalent (the rule call mechanism is viewed as explicit). Besides, each time one of the lazy rules is activated to transform a given proposition from the extension SIG, the *equivalent*() call triggers appropriate rules to produce the equivalent *offsprings*. By this means, the merging of two softgoals triggers operations which recopy, from the extension model, all elements refining the composed softgoal.

3.2. Discussion

Model composition is not considered as a one block operation, but it is decomposed into a succession of steps including the matching and merging phases. A composition specification describes elements that are required to perform such process. In

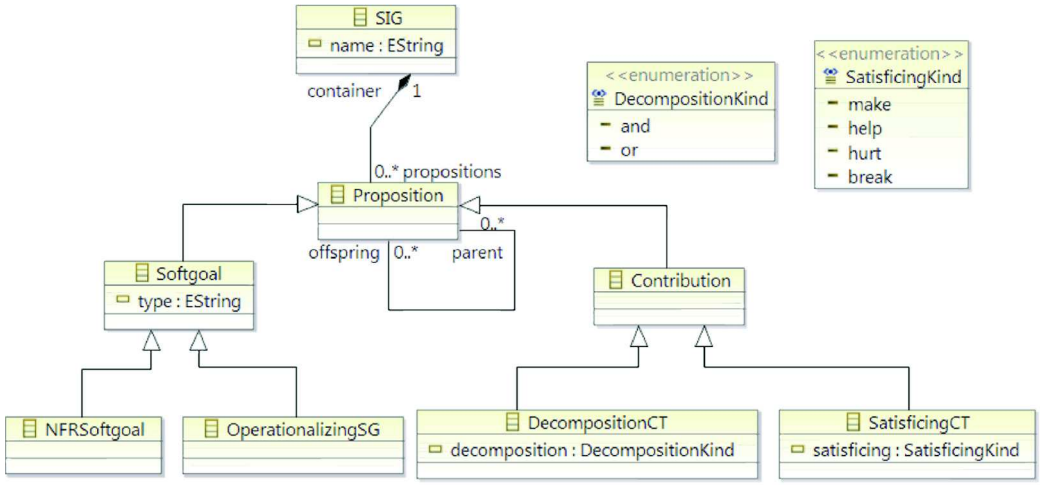

Fig. 6. Abstract syntax for the SIG profile.

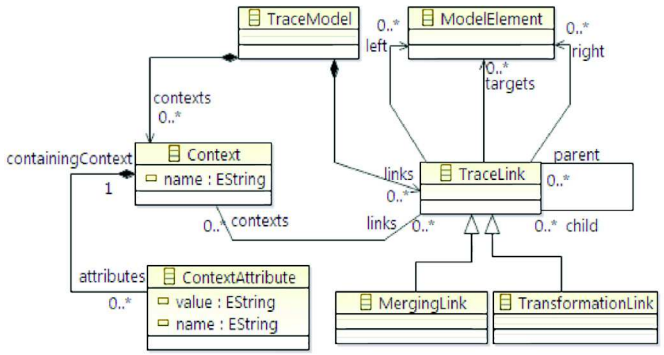

Fig. 7. Composition traceability metamodel.

general case, it defines model elements that should be composed (what to compose), and the way wherein the composition should take place (how to compose) [18,19]. As regards rule based languages like EML [20], the specification takes the form of a set of composition rules. Each one defines a selection mechanism for elements on which the rule should be applied. While the rule body expresses the elementary operations aimed at achieving one of the three possible behaviors (comparison, merging and transformation).

During the execution of a composition specification, its rules are successively applied to the source elements matching their parameters in order to produce the composed model. We call the application of a composition rule to certain source elements "rule activation". Then, keeping track of rule activations is essentially to expose details on the composed model construction. Besides, the specification includes two generative rules (merging and transformation rules) and then elements of the composed model can be divided into two categories. For our illustrative example, the target SIG (Fig. 5) contains elements that originate from only one source SIG (e.g. the "Integrity" softgoal), and those existing in both of them (e.g. the "DataValidation" softgoal). Hence, the trace model must not only express the logical relations existing among the source and target models by capturing traceability links between the constructed elements and the primary ones. Also, the related information has to be categorized on merging and transformation links to enhance traces reusability and expressiveness.

Besides, it is so important to expose details on steps to construct the composed model. For example, the construction of "DataValidation" precedes the "Triggers" softgoal which is added for refining it. Such information can be presented by weaving a relation between traceability links that keep track of each construction. This can be retrieve from rule calls requested during the composition. Two mechanisms for expressing rule calls are available: explicit call that requests the activation of a specific composition rule; and implicit call when a rule activation accesses an element that was created by a previous activation. This latter mechanism is generally allowed by predefined operations that enable the resolution of the target equivalent of certain source elements (e.g. the *equivalent*() operation in EML).

Lastly, in order to give more details about the composition, certain expressive data may be of interest and have to be added to the trace model. As an example, a traceability link can be characterized by the name of the composition rule that produced the referenced target elements.

Nevertheless, existing traceability supports do not allow capturing the composition execution details we just described. Regarding the EML language, the effects of executing a specification are captured by an implicit traceability support [17,21]. Indeed, traces are stored into a trace model that we can access through two mechanisms [17]: exporting the overall model, or exploring traces by invoking the predefined *equivalent*() and *equivalents*() EML operations. However, although the implicit traceability metamodel defines the two types of traceability links we seek to express (merging and transformation links), it does not provide means for nested traces. Moreover, as for any implicit traceability support, the traces generation mechanisms are not configurable to make it possible to capture expressive data that are not defined by the traceability metamodel.

As for explicit traceability supports, we did not find any solution dedicated to the composition of models. Indeed, existing supports focus on the model transformation operation and they disregard, therefore, the composition specificities. For our illustrative example (cf. Section 3.1), all correspondences between elements of the source and composed SIGs will be captured as transformation links without any specialization. However, given that the composition rules express different behaviors (merge and transformation), such representation compromises the traces expressiveness and reusability.

4. Traceability management for model composition

In this section, we present the main concepts related to traceability management in MDE. Thereafter, we introduce a set of re-

quirements that must be met by a traceability approach dedicated to the composition of models.

4.1. Background

Model driven engineering advocates the use of models to represent any artifact handled by the software development process. In this vision, models are used throughout the development process for the expression of requirements, the description of the designed solution and its components, and constitute a testing support. But this intensive use of the modeling technique and the plurality of model representations raise several problems. Among those, model consistency checking, maintenance issues, and model evolution are a few examples. Thus, to deal with such issues, traceability management proves to be promising.

Model transformation traceability refers to the ability to manage relationships between elements handled by an MDD operation. It exposes changes which took place on these model elements and reveals how the source ones contribute in the production of the target models. Thus, model elements represent the main development artifacts to be traced, while transitions from source model elements to target ones are captured by a so called "traceability links". A traceability link is identified as a relationship between one or more source model elements and one or more target model elements [22].

Dirvalos et al. [23] have identified two storage mechanisms for traceability links: intra-model storage and external storage. In the first case, traceability information is retained intrinsically in the source and the target models. However, this approach has several limitations concerning the links management [23]. In the second approach, the traceability information is stored in a separate model. This mechanism facilitates the management of traceability links that are now contained in a dedicated model called "trace model". It may conform to a generic or specific traceability metamodel.

The generic traceability metamodels [8,24] define the basic concepts necessary to the expression of trace models regardless of traceability scenarios (i.e. independently of specific intentions to capture traces and types of elements to be traced). By this means, portability of trace models is supported since all of them conform to the same metamodel. But this is with the detriment of semantic richness aspects. Indeed, the expression of certain information interesting for a given scenario risks not to be allowed by the generic metamodel. Moreover, illegitimate traceability links may be established [23].

On the other hand, specific metamodels [23] allow the expression of semantically rich traceability links. For each scenario, a suitable metamodel is defined taking into account the specificities of the managed models and particularly traceability intentions. Essentially, these metamodels provide support for expressing strongly typified links, and make it possible to define all relevant information to be assigned to them. Nevertheless, the metamodels diversity compromises their interoperability.

4.2. Requirements for a dedicated model composition traceability approach

The scenarios to which our traceability approach is applicable are compositions of heterogonous models. Such scenarios can be specified with different languages (EML, ATL, Kermeta, ...) and involve various types of models (SIG models, class diagrams, BPM models, ...). So, the traceability approach must be generic to deal with this diversity.

Designing specific traceability metamodels reduces the application scope of our proposal since they will not cover all scenarios. Indeed, within our approach the traceability data have to be stored in a separate model that conforms to a generic traceability metamodel. This is to deal with the generic aspect, to support the reuse of traceability links and to reduce the links management effort. On the other side, traceability information must be provided in a form that facilitates its exploitation.

For our specific context, the traceability metamodel has to define two types of links depending on the possible composition rule behaviors: merging links and transformation links (comparison rules will not be traced). Moreover, these traceability links should be related to each other in order to expose the rule calls sequence (explicit and implicit calls). Accordingly, the traces generation mechanism has to address four main concerns:

1. Augment the composition specification with the behavior needed to generate an extra output model (it corresponds to the trace model).
2. For each rule activation, a traceability link is associated therewith (with respect to the composition rule type). This link keeps track of the activation by connecting the concerned source elements with the target elements.
3. Each explicit call to a composition rule (cf. Section 3.2) is traced by weaving a nesting relationship between the link corresponding to the current activation (activation of the calling rule) and the trace link produced by activating the called rule. Thus, the traceability links nesting will be closely modeled on the rule calls sequence.
4. Regarding an implicit call (cf. Section 3.2), a nesting relationship has to be established between the traceability link that keeps track of the current activation and the link referencing the source elements of which the resolution of target equivalents is requested.

In addition, we have set four generic traceability requirements. These latter were derived from an analysis of existing transformation traceability solutions:

5. The traces generation behavior must not be tangled with the primary specification, so as to make it reusable.
6. We aim at reducing the effort to achieve traceability by adopting an automatic and scalable approach. Furthermore, considering the traceability intentions, the traces generation process must be easily configured.
7. The structure of traces must provide an extensibility mechanism. Essentially, this makes it possible to express configurable trace links with regard to the traceability scenario and the specifications of contributing models.
8. A visualization system must be provided to allow expressing trace models in a human friendly presentation.

The following sections detail the core elements of our traceability approach dedicated to the composition of models.

5. Conceptual proposal

In this section, we describe first the metamodel that defines the traceability information to be captured (Section 5.1), and then we detail mechanisms that underpin the production of traces (Section 5.2).

The structuring of traceability information and the generation mechanisms are designed in such a way to satisfy the aforementioned requirements. These latter comprises some requirements that are specific to the model composition operation and generic ones that were derived from the analysis of the main model transformation traceability approaches. The first subset justifies the purpose of our contribution and represents its novelty to consider the model composition specificities. This refers to the elements to be traced, the way traceability information has to be expressed, how

to generate the corresponding traces, and for what traces are captured. While satisfying the generic requirements allows capitalizing benefits of existing works in order to address the traceability challenges [25].

5.1. Composition traceability metamodel

In the literature, several metamodels for model transformation traceability have been proposed [8,26–28]. The base concept is the traceability link (*TraceLink*), which expresses a relationship between a set of source model elements and their equivalents in the resulting model. In the case of the model composition operation, the traceability metamodel must define two types of links: merging links and transformation links (cf. Section 4.2). Such categorization is necessary to consider the composition specificities and to produce expressive and interesting traces.

Thus, trace models produced by the proposed approach conform to our generic traceability metamodel depicted in Fig. 7. The *TraceLink* concept is specialized in *MergingLink* and *TransformationLink*. On one hand, this categorization allows the expression of the composition specific correspondences in a native manner (an alternative is the allocation of additional information to the *TraceLink* concept specifying the link type). On the other hand, it underpins the links reusability (e.g. matching correspondences can be derived directly from merging links). Fig. 11 depicts the graph used to visualize the trace model generated for the illustrative example (cf. Section 6.2.4). This graph was generated using the visualization system which is combined to the traceability metamodel in order to bring greater clarity to its instances (requirement 8, cf. Section 4.2).

A merging link connects two elements that have been combined (they belong to the left and right models) with their correspondent in the composed model. While transformation links keep track of transitions of certain source model elements to the corresponding target elements. We have to notice that transformation links express many-to-many correspondences. Furthermore, an instance of the *TransformationLink* concept may have no source reference (resp. target reference) to allow tracing the creation of new elements (resp. the removal of elements).

We express rule calls by a nesting of traces. Indeed, a rule may invoke another one, and therefore, the traceability link that corresponds to the calling rule (specifically, to the rule activation) must contain the link produced by the called rule. Such a structure can be woven between traces through parent-child relationships. By this means, a multi-scale character is assigned to trace models and the user can explore traces at the granularity level he desires.

Besides, the *Context* concept provides another configuration mechanism. It constitutes the extensibility support which is combined with our generic traceability metamodel (requirement 7, cf. Section 4.2). Depending on the traceability scenario and the implied metamodels, a context allows associating expressive information to traceability links. Basically, the developer has to identify, first, the additional information deemed interesting to express (e.g. the name of the rule that generates the traceability link, types of elements that have been traced, the traceability intention, ...). Thereafter, he defines the context attributes referencing this information. Hence, a context can be viewed as a well thought out combination of attributes, and it can be used to structure a set of traces that share the same contextual information. We have to notice that mechanisms for generating contexts were presented in a previous work [10] and are not within the scope of this paper.

5.2. Traces generation mechanisms

This section presents how we can use the Aspect Oriented Modeling (AOM) and graph transformations techniques to generate traces in an automatic and configurable manner. We first introduce the AOM principles and give a brief overview of graph transformations. After that, we detail the application of graph transformations for implementing the traceability aspect weaving operation.

5.2.1. Generation techniques

We now describe the two techniques that underpin the generation of traceability information: aspect oriented modeling and graph transformations.

Aspect oriented modeling

The Aspect Oriented Software Development (AOSD) [29] provides a paradigm for encapsulating crosscutting concerns of a system (e.g. security, persistence, ...) in appropriate modules, called aspects. This approach relies on the separation of crosscutting and business concerns, and it aims mainly to control the system variability. Besides, AOSD proposes a reusable and configurable evolution support which does not impact the core system.

The use of models being reduced to communication and documentation purposes, AOSD was interested in the productive elements: code artifacts. Thus, several supports for Aspect Oriented Programming (AOP) were initially proposed (AspectJ [30], AspectC++ [31], ...). Afterwards, the large use of models that provides MDE leads to widening the scope of practicing the aspect oriented principles. In this perspective, the aspect definition is no more restricted to programming artifacts, but it concerns any artifact produced in the upstream development phases. Given that these development products are expressed by models, the related aspects are likewise; they are called aspect models.

Thus, the aspect oriented modeling [32] allows the separation of concerns in the earlier phases of the software development process. It results from rising the abstraction levels that portable aspects can be defined. Indeed, their definition is independent from any specific platform, and programming languages which are not supporting aspect orientation could nevertheless be targeted. The definition of an aspect comprises two elements: the model representing the crosscutting concern (the *advice*), and patterns used to capture the application points (*pointcuts*). As for the generation of the global model (expressing all concerns), it is based on the composition of the base model with the aspect model. This composition, usually called aspect weaving, implies two operations: the detection of join points where the aspect will be applied; and the replacement of the identified fragments with the new structure defined in the advice.

Graph transformations

The application of graph transformations [33] relates to several areas in computer science, such as: language definition, functional programming, description of compilers [34], or model transformations within the scope of MDE [35]. This large use is justified by the intuitive character of graph based representations, as well as the maturity of the tool supports (AGG [36], VIATRA [37], ...)

A graph transformation consists in applying specific transformation rules to a source graph. Each transformation specifies the modification of certain parts of this graph by another. The definition of this kind of replacement comprises two parts: a Left Hand Side (LHS) part and a Right Hand Side (RHS) part. The LHS is a graph which is used to determine the source graph's fragments concerned with the application of the rule, while the RHS graph specifies the replacement structure.

Thus, a graph transformation rule p, also known as graph rewriting rule, is a couple of graphs (*LHS, RHS*). Essentially, the application of a rule p on a given graph G comes down [34]:

1. Detecting a subgraph of G isomorphic to the LHS part.
2. Removing elements (nodes and edges) of the subgraph captured by the LHS part which are not present in the RHS part. The resulting graph is called context graph.

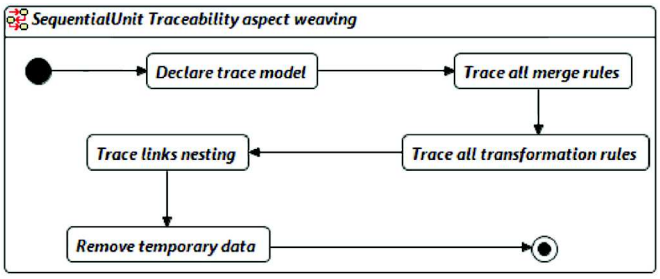

Fig. 8. Graph transformations unit for weaving the traceability aspect.

3. Gluing the RHS graph into the context graph by following the connection points (elements that have been captured by the LHS but remain in the context graph).

5.2.2. Graph transformations for traceability aspect weaving

The aspect oriented principles are aligned perfectly with our objective to generate traces by an automatic and configurable process. The generation concern being encapsulated in a traceability aspect, its application allows the developer to weave, automatically and on demand, the patterns responsible of producing traces (requirements 5 and 6, cf. Section 4.2). Moreover, the aspect definition can be decomposed into tasks specific modules making it possible to configure the application of the global aspect (requirement 6, cf. Section 4.2).

Furthermore, we opted for an AOM approach instead of proposing an AOP based solution that manipulates directly the concrete composition specification. By focusing on the model level, we intend to abstract the specification characteristics, including the concrete syntax nature (textual in EML [11], model based in ATL [12]) and its support for the aspect oriented paradigm (aspect oriented language or not). This will make it possible to define a traceability approach usable in different contexts, and it falls under our goal to build a generic traceability aspect.

However, AOM is presented as a paradigm for encapsulating aspect, at the model level, which requires a mechanism for the implementation of the weaving operation. Two solutions are distinguished: the model composition operation [32] (composing the model that corresponds to the specification to be traced with the traceability aspect model), and the use of graph transformations [38]. In our approach, we opted for the second solution as it supports a better configuration of the aspect application. This advantage is due principally to the following characteristics:

- The modular expression of the traces generation concern by a set of graph transformation rules.
- Monitoring the rules application through application conditions, priorities, and scheduling rules into transformation units.

Moreover, graph transformations allow a perfect simulation of the AOM principles, as follows:

- The traceability aspect corresponds to a graph transformation unit which classifies certain rules in a specific order.
- The LHS part specifies the points where the traceability aspect should be applied (it plays the role of *pointcut*).
- Each RHS part defines the new structure that has to be inserted into the application points captured by the corresponding LHS graph (can be viewed as an *advice*).

The transformation unit depicted in Fig. 8 gives an overview of our traces generation weaving process. This latter addresses the four specific concerns listed in Section 4.2 (requirements 1 to 4). Its first rule declares the trace model like an additional target model of the composition scenario. Thereafter, the following two transformation units are applied, in an iterative way, to trace all merge and transformation rules. We recall that tracing one of these composition rules consists in augmenting it with the behavior needed to produce an appropriate traceability link (merging link or transformation link).

Thus, whenever a composition rule is applied, a traceability link is generated to keep track of its activation. Those links are structured in accordance with the rule calls sequence. This is implemented by two graph transformation rules which are encapsulated in the *"Trace links nesting"* unit (Fig. 8): its first rule corresponds to the implicit call mechanism, while the second one is defined for explicit calls. Finally, the remaining unit removes all the temporary data we use to achieve the weaving operation and which pollute the specification model. The specialization of this weaving process for the EML [11] and ATL [12] languages is detailed in our former works [9,10].

6. Technical solution

6.1. Overview of the tool support

In order to provide a proof of concept for our traceability approach, we have developed an eclipse plug-in called *ComposeTracer* as a tool support. It allows generating and visualizing traces that correspond to the execution of a composition specification. The fact that EMF is a base for a multitude of MDD tools prompted us to choose it as a development context of our plug-in. This is to provide a natural integration of the proposed solution in the basic MDD framework. Hence, *ComposeTracer* can be installed in an eclipse platform integrating EMF and having some dependencies needed to run the building modules of our tool. These are represented by "gears" in Fig. 9.

- Serialization module: It gathers a set of parsers that are specific to the supported composition languages (we focus on EML within this paper). The parser allows producing the model corresponding to the concrete specification to trace (textual in the case of EML). In addition, it makes it possible to realize the reverse operation by transforming any model that conforms to the EML abstract syntax to the executable specification (expressed by the EML concrete syntax).
- Weaving module: It constitutes the core component of our tool support, and it allows weaving the traceability aspect. Essentially, this module is developed in the interest of applying the graph transformation units which were evoked in Section 5.2.2 on the model resulting from the serialization of the composition specification.
- Composition module: In order to generate traces, our approach uses the same construction mechanisms that produce the composed model. If the weaving module adds the behavior related to the traces generation concern, the resulting specification must be executed to produce the expected models (the composed model and the trace model).
- Visualization module: The proposal of a visualization system is a paramount requirement for traceability supports. Its objective is to improve the readability of trace models with a view of reasoning about possible exploitations of those. The proposed visualization module meets this need, and it is based on a graph placement algorithm defined around our traceability metamodel.

In what follows, we give some implementation details of each module.

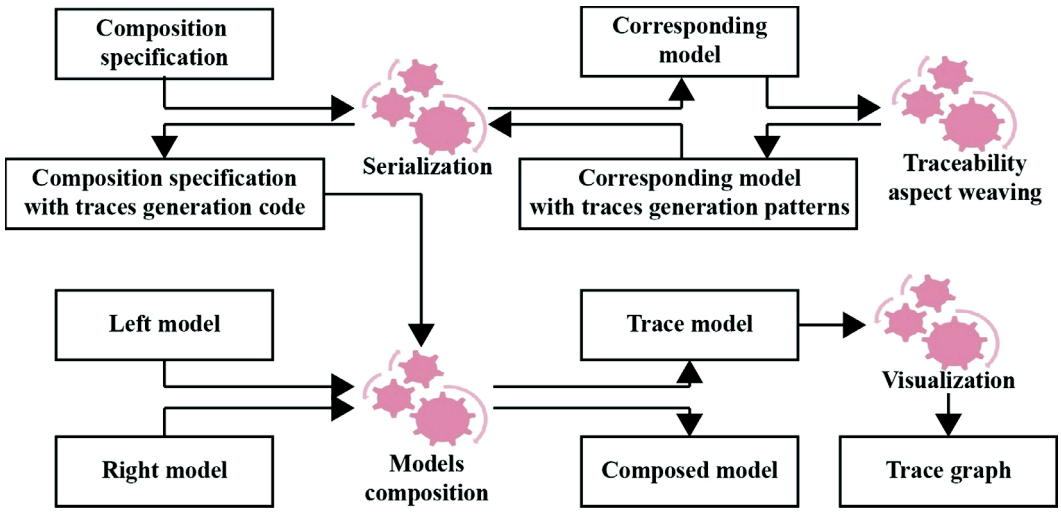

Fig. 9. Global architecture of the *ComposeTracer* plug-in.

6.2. Implementation details

6.2.1. Serialization module

The current versions of the languages provided by the Epsilon platform [20] are not based on EMF metamodels. They rely on ANTLR[1] grammars to produce lexical and syntactic analyzers for converting textual specifications to Abstract Syntax Trees (ASTs). Consequently, the Epsilon platform does not include any serialization tool for concrete specifications towards models. Nonetheless, it proposes, in its official documentation [20], metamodels associated with some of the supported languages (EOL, ETL, EML, etc.). These proposals will constitute a basis for the definition of metamodel based languages pending a future amendment of runtimes manipulating ASTs to the use of model elements.

Knowing this, we were constrained to implement a serialization tool for EML. The proposed solution has been developed through the EMFText[2] project while relying on the abstract syntax defined for the EML language [20]. EMFText is a tool to describe correspondences between elements of an abstract syntax expressed by an Ecore metamodel and their equivalents in a concrete syntax specified using a formal grammar. It provides a complete infrastructure associated with the language, including parsers and integrated editors in the Eclipse platform.

For each concept defined in the EML abstract syntax, an EMFText rule is defined. The latter specifies the textual representation in which an instance of the concept will appear. This syntax is derived from the EML grammar, and it is described in a text file based on EBNF [39].

Once the EML metamodel and the EMFText rules associated therewith are specified, the corresponding serialization infrastructure can be automatically generated. Indeed, EMFText integrates a code generation module that derives a set of plug-ins. Of these, we were interested in the components responsible of loading and saving EML models. We use them to implement two operations that serve as text-to-model and model-to-text transformations and allow serializing EML specifications to EMF models and back again.

6.2.2. Weaving module

In Section 5.2.2, we presented the weaving process of our traceability aspect. We recall that specialization details of this process for the EML language are not within the scope of this paper and they were already presented in a previous work [9]. Basically, the traces generation behavior is expressed by a set of graph transformations implemented with Henshin [40] and scheduled in a sequential unit. The latter can be applied to the model corresponding to a EML specification.

```
1  rule TransformNFRSoftgoal
2  transform l : m1!NFRSoftgoal
3  to t : m3!NFRSoftgoal , tr : trace!TransformationLink
4  {
5          t.type = l.type;
6          t.container = sig;
7          traceModel.links.add(tr);
8          tr.left.add(l);
9          tr.targets.add(t);
10         resolvedElt = l.offspring.equivalent();
11         t.offspring = resolvedElt.select(it | not it.isKindOf(
               trace!TraceLink));
12         tr.child.addAll(resolvedElt.select(it | it.isKindOf(trace
               !TraceLink)));
13 }
```

Listing 1. Example of a composition rule with traces generation code.

Listing 1 illustrates an excerpt of the specification resulting from the application of our weaving process on the composition module that is presented in Section 3.1.3. Given that the *TransformNFRSoftgoal* rule expresses a transformation behavior, a traceability parameter of type *TransformationLink* is declared as another target parameter (Listing 1, line 3). Besides, the traceability information is assigned to it by initializing the left and target properties with references of the source and target elements (they correspond to values that take the l and t parameters while a given rule activation).

In addition, the call to the *equivalent*() operation has been captured and replaced with the fragment that divides its return into trace model elements and default target elements (Listing 1, lines 10–12). The first sub-set is used to perform the original call to the *equivalent*() operation (Listing 1, line 11), while traceability elements are assigned as children of the current trace link (Listing 1, line 12). By this means, we nest traces depending on the rule calls sequence.

6.2.3. Composition module

Recalling from Section 3.1.3, a composition specification is expressed in the Epsilon platform by two separated scripts: the comparison module, and the merging module. Likewise, the execution follows the typical composition process and proceeds in two steps: the matching phase, and the composition (merging) phase. Indeed, correspondences are first identified and captured in an implicit traceability model. This latter is then exploited, during the execu-

[1] ANother Tool for Language Recognition http://www.antlr.org/.
[2] http://www.emftext.org/index.php/EMFText.

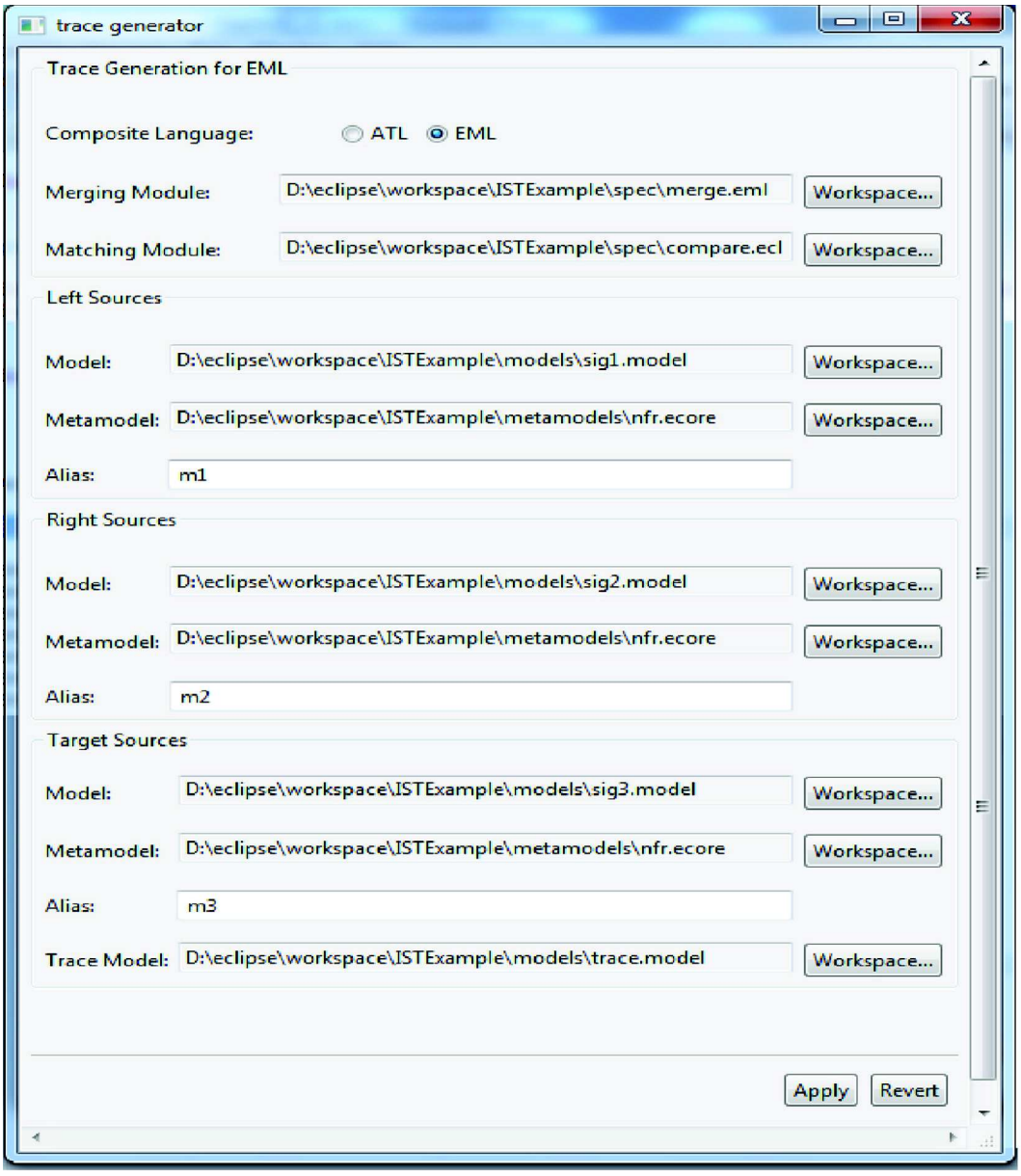

Fig. 10. GUI to launch an EML specification.

tion of the merging module, to switch the application of declarative rules and to resolve target equivalents.

In order to programmatically execute these two modules, we reuse the EspilonStandalone[3] class that includes a set of facilities for launching Epsilon scripts. However, some operations need to be redefined to specify the launch parameters and target the appropriate engine for our particular context (the EML and ECL engines). Also, some properties concerning the normal composition scenario (path of the composition module, source models, storage location for the composed model, implied metamodels packages, etc.) as well as the storage location for the trace model are required. A suitable view allows users to set these arguments needed to perform the composition. This plug-in interface appears when launching the traces generation process (see Fig. 10).

6.2.4. Visualization module

The basic building block of our visualization module is a model-to-text transformation that manipulates the implied models (source, composed, and trace models) for the purpose of producing a single text script written in DOT [41]. The latter is then interpreted by the placement algorithm of Graphviz [42] to draw the corresponding graph. The aim is to provide a human friendly representation for trace models that makes easier their examination.

The proposed model-to-text transformation can be applied to any trace model produced by our plug-in as it ignores the definition of the implied metamodels (to which conform the source and composed models). Essentially, traceability links are represented by nodes whose identifiers are used for the construction of possible edges connected with (traceability nesting relationships in which they participate). In addition, for each concept defined in our traceability metamodel, we match a characterizing display mode:

[3] http://www.eclipse.org/epsilon/examples/index.php?example=org.eclipse.epsilon.examples.standalone.

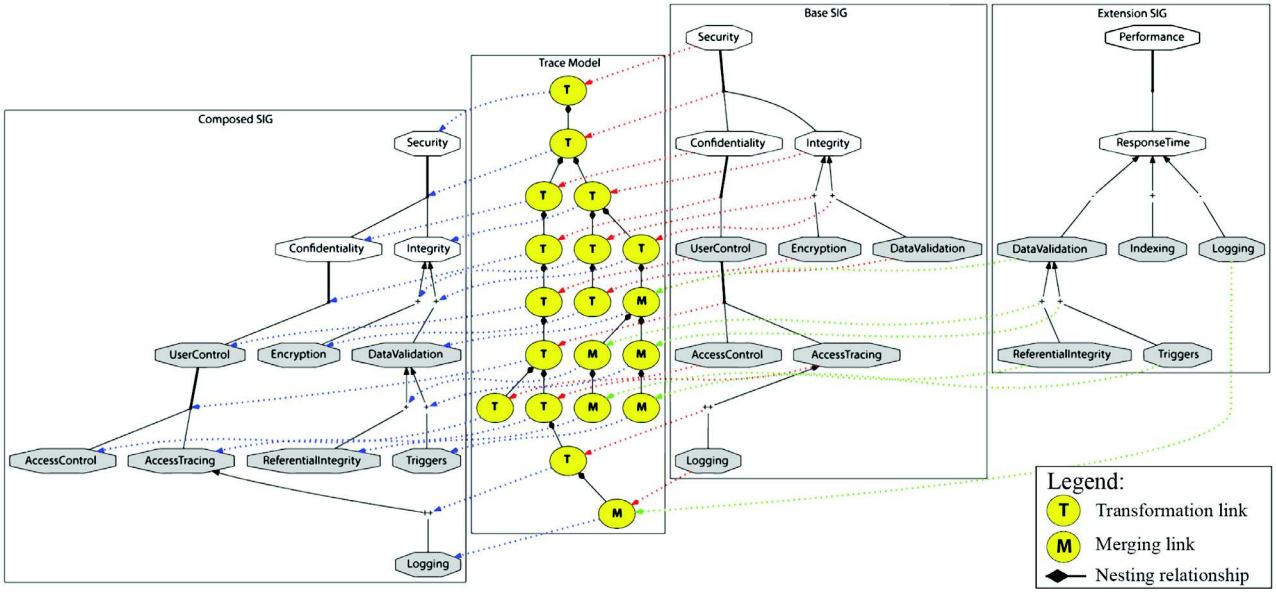

Fig. 11. Trace model corresponding to the SIGs composition scenario.

- The trace model is represented by a subgraph comprising all its traceability elements (traceability links, nesting relationships, contexts).
- Merging links correspond to yellow nodes labeled with 'M'.
- Transformation links are represented by a similar mode, except for the prefix 'T' which characterizes them.
- A traceability nesting relationship corresponds to a solid line connecting the parent link to its child. A diamond is added to the edge-end referencing the parent link.
- The *left*, *right*, and *target* references of traceability links are represented by dashed lines.
- Any element belonging to the source or composed models is represented by a simple node labeled with the name of its classifier and the value taken by the first property. Furthermore, the corresponding node is arranged inside the subgraph representing the container model.

Regarding the last point, we chose to represent all elements of the handled models instead of being limited to those referenced by the captured traces. Although this choice increases dimensions of the resulting graph, but allows having a better visibility on the execution details and reasoning about any inconsistency or incompleteness.

Besides, the generic character of our visualization system compromises the expressiveness of the overall representation. Indeed, the display mode associated with a managed model is too simple to expose all information it contains and to reveal the intrinsic logical relations. The ideal would be to set up the system in order to represent each model by its concrete syntax. As for the illustrative example (cf. Section 3.1), we have configured our visualization system for the SIG profile. The trace model generated for this scenario is depicted in Fig. 11.

7. Validation

After presenting the implementation details, we introduce in this section a set of points for the validation of the proposed approach. The main objectives are:

- To assess the relevance of the generated traces by means of some appropriate metrics (Traces relevance).
- To discuss the extent to which our approach fits to the increase of elements to be traced (Scalability).

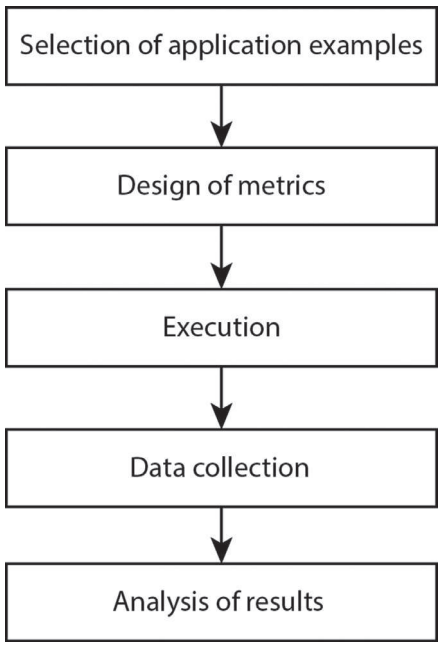

Fig. 12. Validation process.

The validation follows a process inspired from Vara et al. [28]. It includes five steps: selection of application examples, design of metrics, execution, data collection and analysis of results (Fig. 12).

7.1. Traces relevance

7.1.1. Selection of application examples

In order to assess the traces relevance, three composition scenarios were selected:

- **Scenario 1**: Perform the composition scenario presented as an illustrative example (cf. Section 3.1).
- **Scenario 2**: Employ the specification we introduced in Section 3.1.3 to realize the composition scenario presented by Wei et al. [14]. The selected SIG models are the decompositions of "Friendliness" (composed version) and "Security".

- **Scenario 3**: Use the integrating live reports example which is described in the Interactive Television Applications (TVApp) case study[4]. The selected scenario consists in integrating a live report in the *ChampionsLeague* TVApp model. The composition specification as well as the involved models can be downloaded from the Epsilon GIT.

On the one side, scenarios 1 and 2 were selected to evaluate changes in traces relevance following the increase of models size. On the other side, the last scenario allows us to apply our traceability approach to a pre-existing specification.

7.1.2. Design of metrics

The relevance of a trace model reflects the extent to which information it includes is complete and does not present any unwanted information (noise). This qualitative aspect is very present in the information retrieval field. In our particular context, two research questions were posed to assess the traces relevance:

1. Does the trace model keep track of all rule activations constructing the composed model?
2. Are all composition rule calls traced by traceability nesting relationships?

Accordingly, we have defined two metrics to assess the relevance of a trace model. The first one is related to the traceability of the Composed Model Construction (*CMC*). Each element of the composed model must be connected to a traceability link which keeps track of its construction. The *CMC* metric provides an idea about the percentage of target elements that are traced. It computes the ratio between the number of traceability links and the number of the composed model elements. While the second metric addresses the Composed Model Structuring (*CMS*) by providing the rate of tracing rule calls. The idea is that each relation between two target elements is woven through a rule call. This latter is performed between the two rule activations which produce the composed elements connected with the relation. The *CMS* computes the ratio between the numbers of traceability nesting relationships and the composed model relations.

Definition 1.

$$CMC = \frac{|L_{tr}|}{|N_{M_c}|}$$

Definition 2.

$$CMS = \frac{|R_{tr}|}{|E_{M_c}|}$$

These metrics are defined based on the following notations. A trace model (M_{trace}) is viewed as a set of traceability links (L_{tr}) and a set of traceability nesting relationships (R_{tr}). Furthermore, a model is generally defined by a graph [43]. We draw on this definition, and we note by:

- N_{M_c}: All elements (nodes) of a composed model M_c (e.g. classes, attributes, softgoals, etc.).
- E_{M_c}: All relations (arcs) of a composed model M_c. They describe nodes connections (e.g. an attribute belonging to its owned class).

For the metrics *CMC* and *CMS*, a value greater than 1 reflects that the trace model comprises unwanted information. While a value less than 1 can determine a possible incompleteness. Nevertheless, the metrics *CMC* and *CMS* do not constitute, alone, an undeniable base for relevance assessment. For example, a transformation rule may construct two target elements upon a single activation. In this case, the corresponding trace model may be relevant while giving a value less than 1 for *CMC* (the rule activation will be captured by a single traceability link). Thus, we define, in addition to the metrics *CMC* and *CMS*, two other metrics that focus on the coverage rate of target elements by traces. The first one (CMC_1) presents the rate of the composed model elements whose construction is captured by one traceability link ($|N_{M_{c1}}|$), while the second metric (CMC_*) concerns those related to multiple traceability links ($|N_{M_{c*}}|$).

Definition 3.

$$CMC_1 = \frac{|N_{M_{c1}}|}{|N_{M_c}|}$$

Definition 4.

$$CMC_* = \frac{|N_{M_{c*}}|}{|N_{M_c}|}$$

7.1.3. Execution

We started the execution step by developing the specification we used to composed SIG models (cf. Section 3.1.3) and creating the managed models corresponding to scenarios 1 and 2. For scenario 3, both the composition specification and the involved models were downloaded from the Epsilon Git. Thereafter, using an eclipse instance, we launch these specifications using our *Compose-Tracer* plugin in order to produce the composed and trace models.

7.1.4. Data collection

To be able to measure the defined relevance metrics, we calculated the total number of elements ($|N_{M_c}|$) and relations ($|E_{M_c}|$) belonging to the composed model. Then, we examined the generated trace model to determine the number of the traceability links ($|L_{tr}|$), the number of the traceability nesting relationships ($|R_{tr}|$), and the numbers of the composed model elements that are referenced by only one traceability link ($|N_{M_{c1}}|$) and those connected with more than one traceability link ($|N_{M_{c*}}|$). The following table summarizes the obtained results (cf. Table 1).

Table 1
Collected data for the selected application examples.

	$\|N_{M_c}\|$	$\|E_{M_c}\|$	$\|L_{tr}\|$	$\|R_{tr}\|$	$\|N_{M_{c1}}\|$	$\|N_{M_{c*}}\|$
Scenario 1	20	37	19	18	19	00
Scenario 2	31	63	30	31	30	00
Scenario 3	20	19	19	18	19	00

Table 2
Relevance metrics for the selected application examples.

	CMC	*CMS*	CMC_1	CMC_*
Scenario 1	0,95	0,48	0,95	00
Scenario 2	0,96	0,49	0,96	00
Scenario 3	0,95	0,94	0,95	00

7.1.5. Analysis of results

The results analysis is in accordance with the questions posed during the design step:

1. Does the trace model keep track of all rule activations constructing the composed model?

We remark that for these scenarios, the construction of more than 95% of the target elements were captured by trace models (cf. Table 2, Column 3). Besides, given that all values of CMC_* are zero, we can deduce that the generated trace models do not include redundant links (cf. Table 2, Column 4). Moreover, values of the metric *CMC* are less than 1 (cf. Table 2, Column 1), and they

[4] http://www.dsmforum.org/events/MDD-TIF07/InteractiveTVApps.pdf.

do not imply the existence of noise in the generated trace models. Indeed, the obtained values are very close to 1.

2. Are all composition rule calls traced by traceability nesting relationships?

The results summarized in Table 2 show that for the two first scenarios, traceability nesting relationships keep track of more than 48% of rule calls (cf. Table 2, Column 2). An exception is the third scenario for which a very high value is associated to the metric *CMS*.

As for scenarios 1 and 2 that exploit the same specification, values of the four relevance metrics are very close although these scenarios involved models of different sizes. This therefore shows a dependence between the traces generation mechanisms and the style used to express the specification. Indeed, following a thorough analysis of the selected specifications, we have drawn the following conclusions:

- It is advisable to define a specific composition/transformation rule for each target metamodel classifier. This rule will construct exclusively the classifier instances.
- The metric *CMS* is very dependent on the use of rule call mechanisms that are provided by the language (e.g. the *equivalent*() and *equivalents*() operations in Epsilon task languages [20]). We encourage, therefore, the use of these operations instead of directly weaving structural relationships between target model elements.

7.2. Scalability

7.2.1. Selection of application examples

We have chosen five composition scenarios to assess the scalability of our traceability support. These scenarios apply the specification that allows composing SIGs (cf. Section 3.1.3) to various models of different sizes.

7.2.2. Design of metrics

Scalability is one of the most crucial aspects to be covered by a software solution. Its importance is being accentuated as the traces generation concern remains secondary to the production of the composed model. Therefore, delay in response should not be felt by the developer.

For evaluating scalability of our traceability support, we relies on the assessment of the response time. It measures the difference between the time of generating the composed model alone (T_{M_c}) and the time that takes the realization of the same composition scenario while producing a trace model ($T_{M_c+M_{trace}}$). This difference (Δ_{trace}) provides an overview of the time required for weaving the traceability aspect and constructing the trace model.

7.2.3. Execution

The first step was to create the SIG models that are involved in the selected scenarios. For each case, we used our *Compose-Tracer* plug-in to perform the composition scenario with and without trace generation. We ran our experiments on the same eclipse instance and the *JVM monitor* was chosen for measuring the execution time.

7.2.4. Data collection

For each composition scenario, we measured the time required for generation the composed model alone ((T_{M_c})), and execution delay for executing the same scenario while enabling the production of traces (($T_{M_c+M_{trace}}$)). We noted also the number of the composed model elements (($|M_c|$). The following table summarizes the obtained results (cf. Table 3).

Table 3
Δ_{trace} to support traces generation (milliseconds).

$\|M_c\|$	$T_{M_c}(ms)$	$T_{M_c+M_{trace}}(ms)$	$\Delta_{trace}(ms)$
09	47	218	171
18	64	141	177
29	72	250	178
65	78	265	187
104	86	281	195

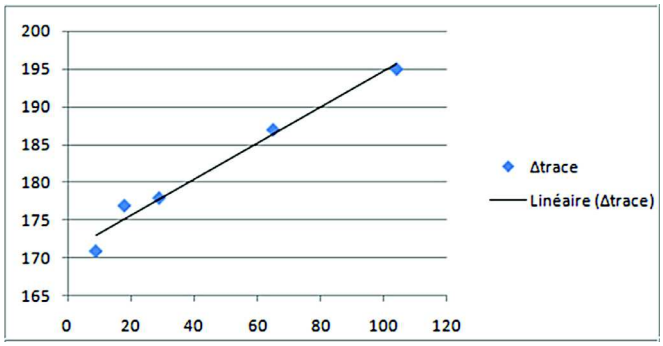

Fig. 13. A scatter plot of the collected data.

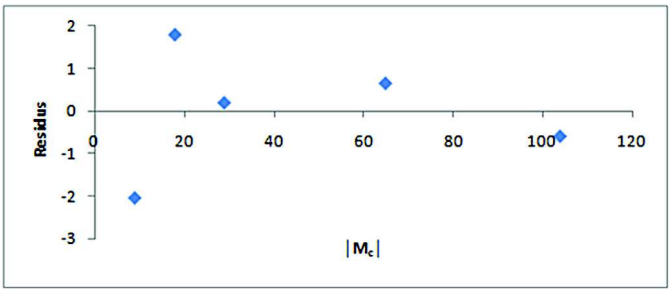

Fig. 14. Residual plot.

Table 4
Results of univariate linear regression.

Coefficient	0,237
Constant	170,9
p-value	0,001
R^2	0,976

7.2.5. Analysis of results

In this section, we describe the statistical analyses we performed to discover the relationship between the values of Δ_{trace} and the number of the composed model elements ($|M_c|$). We employ a univariate regression method as we have determined one explanatory variable (i.e. number of the composed model elements) and the dependent variable is Δ_{trace}.

The Fig. 13 presents original values for both variables $|M_c|$ and Δ_{trace} as well as the corresponding regression line which reveals a linear distribution. Also, the points in the residual plot are randomly dispersed around the horizontal axis (see Fig. 14). Accordingly, the regression line seems to be quite a good fit to the data.

So we applied a univariate linear regression analysis (see Table 4). For the value of coefficient of determination we obtained $R^2 = 0.976$ which means that 97% of a whole variance is explained by the model. Besides, the correlation is significant (p-value < 0.005) and it is very close to 1, which means there is a strong linear association between $|M_c|$ and Δ_{trace}. Accordingly, we can affirm that our traceability support fits to the increase of elements to be traced in comparison with other nonlinear regression models (power or exponential regression). Indeed, the delay for producing traces remains propositional to composed model size.

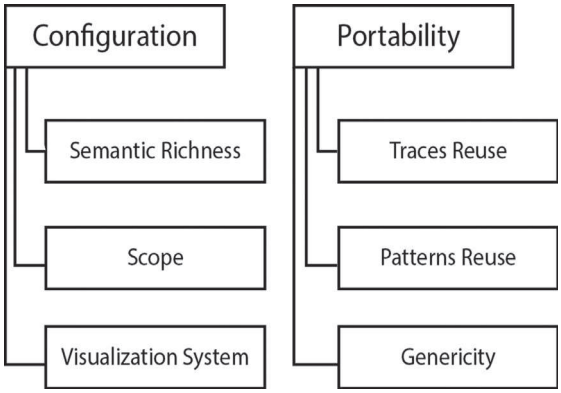

Fig. 15. Refinement of configuration and portability challenges.

8. Discussion

In what precedes, we presented the implementation of the *ComposeTracer* plug-in. This prototype served as a support to the realization of some experiments that have led to validate our approach. Indeed, we could assess the relevance of the generated traces and the scalability of the traceability support.

In addition to scalability, we are also interested in two other significant traceability challenges [25]: configuration and portability. But since we have not validated experimentally the satisfaction of these last criteria, this section discusses the contribution of our approach to overcome them. For these challenges, we have relied on descriptions proposed by the members of the center of excellence for software traceability [25] while providing definitions appropriated to the context of model transformations. Indeed, we have established a specific set of criteria organized into the retained challenges (Fig. 15):

- **Configuration**: It specifies the ability of the approach to capture and to provide valuable information for a given traceability scenario while supporting changes in user needs. In our analysis, we link this challenge to three sub-criteria. The first is the extent to which the approach captures any expressive traceability information (semantic richness). The second aspect identifies the possibility of choosing a subset of elements to trace (the scope configuration). The last criterion is related to the existence of a visualization system that makes explicit the representation of the trace model.
- **Portability**: It considers two types of reusable information: the trace model and the traces generation patterns. Indeed, the way the traceability links are structured and stored directly affects their reusability. In addition, the generation patterns can be loosely coupled to a specification or strongly dependent of it. The last portability aspect to consider is the applicability of the approach to various composition/transformation supports (the generic aspect).

8.1. Configuration

The generated trace model has the advantage of being both relevant and semantically rich (it is possible to capture any information deemed useful by the developer via the *Context* concept). In addition, it does not only keep track of correspondences between the source and target model elements, but traceability nesting relationships are woven between traceability links to expose the rules' activations sequence. However, we noted in Section 7.1.5 that many nesting relationships escape the traceability support (an average of 37% for the realized experiments). This is due to the traces generation mechanism which exclusively captures the rule calls apparent in the specification. As a solution, an adaptation is envisaged for our traceability support to analyze other traceable operations, and therefore generate more fine grained traces.

The selection of model elements to be traced (the scope configuration) is another configuration aspect that has guided, from the outset, the design of our traceability approach. Indeed, the choice of graph transformations to implement the traceability aspect is based on their modularity and application control. Accordingly, the weaving of the traceability aspect can, theoretically, be parameterized according to the traceability scenario. For example, we can only keep track of the merging rules' activations by restricting the application of the aspect to the related transformation unit.

Also, we provide support for a post-generation configuration. Indeed, our traceability support allows generating a trace model that covers all possible configurations. Thereafter, the developer can choose a sub-set of interesting traces by specifying a selection condition for them (e.g. to be a merging link, to reference a specific model element type, etc.). This condition constitutes the input of a reduction algorithm that we have implemented to extract the interesting links. Besides, the visualization system represents the selected ones in a reduced graph.

8.2. Portability

The generated traces are stored in an external model that conforms to a generic traceability metamodel. By this means, traceability information is reusable in different contexts. In addition, trace models can be handled by generic transformations without requiring to be adjusted (e.g. the transformation that generates the DOT script used to visualize traces).

On the other side, the aspect-oriented modeling techniques allow us to encapsulate the traces generation concern in reusable patterns. Indeed, the graph transformations unit that implements the traceability aspect [9] is applicable to any composition specification written in EML. Moreover, even composition languages that do not support the aspect orientation can be targeted through the completion of the weaving operation in a high-level of abstraction.

However, this portability that characterizes the traces generation support remains dependent of the language used to express the composition. Indeed, the traceability weaving unit is defined around the concepts that propose this latter. Given this limitation, we are examining another solution which relies on a generic composition language to accomplish the weaving [44].

9. Related work

This section briefly presents related works that fit under the transformation traceability field. We recall that the expressiveness of the generated traces is compromised because existing works consider the composition of models as a specific case of model transformations. Indeed, no relevant concept is defined for the merging behavior. The workaround is the definition of the corresponding expressive data for specifying the link type, but this supposes that the traceability support provides a extensibility mechanism for the structuring of traces. On the other side, the flexibility of the traces generation support determines if such extensibility is possible (it is possible to specify how to capture the new data), and its portability expands the application scope. The overview that follows focuses on techniques used to build existing traces generation supports.

The work proposed by Jouault [8] is the basis for several approaches addressing the transformation traceability management. The main idea is to derive traceability information from inter-metamodels relationships that are expressed in the transformation specification. For that end, the fragments responsible of generating traces are included in the specification. In addition, this lat-

ter is viewed as an artifact that can be manipulated automatically for achieving this concern. Indeed, the proposed solution uses a Higher-Order Transformation (HOT) that can be applied to trace ATL transformations.

Another group of traceability approaches rely on aspect-oriented principles to address traceability management issues (essentially, the diversity of transformation languages and the imperative character of some of them). The principle is to consider the traces generation as a crosscutting concern that has to be encapsulated in a traceability aspect. Just as the use of HOTs, the application of this aspect allows weaving the traces generation patterns into the transformation specification. Besides, the main advantage of adopting such approach is to be able to configure the aspect application. Within this scope, Amar et al. [26] propose an AOP based approach to trace Java/EMF transformations. And Grammel and kastenholz [27] define a support to extend various transformation languages (i.e. QVT[5] and oAW[6]) with a traceability mechanism. The authors describe two mechanisms for generating traces: use AOP to address this task in the absence of an implicit traceability support, and transform the implicit trace model to conform to their generic metamodel otherwise.

The proposal of Grammel and kastenholz [27] is presented as a generic traceability support that allows augmenting different transformation languages with a traceability mechanism. However, it can be viewed as a gathering of various language specific traceability supports. On the other side, the work presented by Vara et al. [28] is obviously distinguished by its generic character. Indeed, the extraction of traceability relationships is implemented by a HOT operating on the high-level specifications of transformations that are independent from a given language. These specifications are then transformed to produce low-level specifications and the executable modules afterwards. The execution of these latter allows generating target models accompanied with the corresponding trace models. However, such approach is applicable when developing new transformations and it cannot be used to trace existing ones.

Furthermore, several traceability approaches based on TGGs (Triple Graph Grammars) [45] were proposed in the literature (e.g. [46,47]). TGGs provide means for specifying graph transformations that allow implicit generation of traceability links. Indeed, a TGG rule consists of a source graph, a target graph and a correspondence graph. This latter can be viewed as a set of traceability links between corresponding elements belonging to the source and target graphs [48].

Our proposal takes advantages of existing approaches to provide an automatic, configurable and portable traces generation support. Indeed, a HOT allows handling abstract specifications to insert the traces generation patterns. However, it constitutes a one block and does not present ways to configure its application. Against this issue, the aspect definition is characterized by its modularity, but the use of AOP is restricted to textual specifications expressed on languages that primary support the aspect orientation. Accordingly, instead of adopting an AOP based approach, we have chosen to rely on an AOM solution implemented by graph transformations. By this means, any language could be targeted regardless its features (concrete syntax nature, its support for aspect orientation, etc.).

10. Conclusion and future work

Although traceability management issues in MDE are widely studied, and that many related contributions have been proposed, we did not find any traceability solution dedicated to the model composition operation. The only possible option would be to apply the existing transformation traceability supports to trace composition scenarios. Nevertheless, exploiting such solutions leads to expressiveness and reusability problems.

To address these limitations, our approach can automatically generate traces of a rule based composition (the EML and ATL languages are currently supported). The principle is to transform the composition specification in a version designed to produce, in addition to the expected composed model, a trace model. For that, the generation of this extra output is encapsulated in a traceability aspect implemented by a set of graph transformation rules. As for the traces structuring, we have defined a traceability metamodel that expresses the relationships kinds to be captured in a model composition scenario. Additionally, this generic metamodel is augmented with an extensibility mechanism for introducing traces semantics.

Our traceability approach has been implemented in an Eclipse plug-in "*ComposeTracer*". In order to validate the proposal, the composition of SIG models has been selected as an illustrative example. The corresponding trace models produced by the tool support have served to assess scalability and the relevance of traces.

Finally, the presented work opens up many perspectives. We divide the future work into two main areas: traces generation and traces exploitation. Experiments that we have conducted to assess the traces relevance show that we capture a large part of traces exposing the composed model construction, but the weaving of structural relations is not well traced. Therefore, it seems interesting to extend the traceability aspect to ensure a deeper analysis of the composition specification and gather more fine grained and complete traces. In the same context, we intend to target other composition languages.

Regarding the traces exploitation, we have experienced how to use traceability information to verify the composed model correctness. However, the analysis is completely manual and no support is available to guide the developer. To address this issue, we are exploring the definition of heuristics on the compliance of a trace model to constraints specified dynamically by the developer (e.g. all instances of a specific source type must be referenced by a traceability link, the trace model must include one root traceability link, etc.). A list of possible causes for non-conformity must be proposed to reveal errors introduced in the specification or inconsistency of source models.

These models being doomed to evolve, the use of traces for analyzing the impact of changes is a widely recognized practice. Accordingly, we are planning to develop a support that allows detecting changes and propagating their effects in the implied models. Moreover, this support must preserve the trace model consistency by ensuring a permanent maintenance of traces.

Appendix A

This section presents the specification that allows composing softgoal trees. It includes the comparison module written in ECL (cf. Listing 2) and the merging module specified in EML (cf. Listing 3).

```
rule MatchNFRSoftgoals
        match l : m1!NFRSoftgoal
        with r : m2!NFRSoftgoal {
        compare {
                return l.type==r.type;}
}

rule MatchOperationalizingSGs
        match l : m1!OperationalizingSG
        with r : m2!OperationalizingSG {
        compare {
                return l.type==r.type;}
}
```

Listing 2. ECL specification for comparing SIG models.

[5] Query View Transformation. http://www.omg.org/spec/QVT/.
[6] openArchitectureWare. http://www.openarchitectureware.org/.

```
pre CreateSIG{
var sig = new m3!SIG;
sig.name='Global SIG';
}

rule MergeNFRSoftgoals
        merge l : m1!NFRSoftgoal
        with r : m2!NFRSoftgoal
        into t : m3!NFRSoftgoal
        {
        t.type=l.type;
        t.container=sig;
        t.offspring=l.offspring.includingAll(r.offspring).
             equivalent();
        }

rule MergeOperationalizingSGs
        merge l : m1!OperationalizingSG
        with r : m2!OperationalizingSG
        into t : m3!OperationalizingSG
        {
        t.type=l.type;
        t.container=sig;
        t.offspring=l.offspring.includingAll(r.offspring).
             equivalent();
        }

rule TransformNFRSoftgoal
        transform l : m1!NFRSoftgoal
        to t : m3!NFRSoftgoal {
        t.type = l.type;
        t.container=sig;
        t.offspring=l.offspring.equivalent();
}

rule TransformOperationalizingSG
        transform l : m1!OperationalizingSG
        to t : m3!OperationalizingSG {
        t.type = l.type;
        t.container=sig;
        t.offspring=l.offspring.equivalent();
}

rule TransformDecompositionCT
        transform l : m1!DecompositionCT
        to t : m3!DecompositionCT {
        t.decomposition=l.decomposition;
        t.container=sig;
        t.offspring=l.offspring.equivalent();
}

rule TransformSatisficingCT
        transform l : m1!SatisficingCT
        to t : m3!SatisficingCT{
        t.satisficing=l.satisficing;
        t.container=sig;
        t.offspring=l.offspring.equivalent();
}

@lazy
rule TransformNFRSoftgoal1
        transform l : m2!NFRSoftgoal
        to t : m3!NFRSoftgoal {
        t.type = l.type;
        t.container=sig;
        t.offspring=l.offspring.equivalent();
}

@lazy
rule TransformOperationalizingSG1
        transform l : m2!OperationalizingSG
        to t : m3!OperationalizingSG {
        t.type = l.type;
        t.container=sig;
        t.offspring=l.offspring.equivalent();
}

@lazy
rule TransformDecompositionCT1
        transform l : m2!DecompositionCT
        to t : m3!DecompositionCT {
        t.decomposition=l.decomposition;
        t.container=sig;
        t.offspring=l.offspring.equivalent();
}

@lazy
rule TransformSatisficingCT1
        transform l : m2!SatisficingCT
        to t : m3!SatisficingCT{
        t.satisficing=l.satisficing;
        t.container=sig;
        t.offspring=l.offspring.equivalent();}
```

Listing 3. EML specification for composition SIG models.

References

[1] G. Spanoudakis, A. Zisman, Software traceability: a roadmap, in: Handbook of Software Engineering and Knowledge Engineering, World Scientific Publishing, 2004, pp. 395–428.

[2] S. Walderhaug, E. Stav, U. Johansen, G.K. Olsen, Traceability in model-driven software development, Des. Softw. Intensive Syst. (2008) 133–159.

[3] J. Cleland-Huang, C.K. Chang, M.J. Christensen, Event-based traceability for managing evolutionary change, IEEE Trans. Softw. Eng. 29 (9) (2003) 796–810, doi:10.1109/TSE.2003.1232285.

[4] A. von Knethen, M. Grund, Quatrace: A tool environment for (semi-) automatic impact analysis based on traces, in: 19th International Conference on Software Maintenance (ICSM 2003), 2003, pp. 246–255, doi:10.1109/ICSM.2003.1235427.

[5] B. Amar, H. Leblanc, B. Coulette, P. Dhaussy, Trace transformation reuse to guide co-evolution of models, in: ICSOFT 2010 - Proceedings of the Fifth International Conference on Software and Data Technologies, Volume 2, 2010, pp. 73–81.

[6] N. Aizenbud-Reshef, B.T. Nolan, J. Rubin, Y. Shaham-Gafni, Model traceability, IBM Syst. J. 45 (3) (2006) 515–526, doi:10.1147/sj.453.0515.

[7] I. Galvão, A. Goknil, Survey of traceability approaches in model-driven engineering, in: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007, 2007, pp. 313–326, doi:10.1109/EDOC.2007.42.

[8] F. Jouault, Loosely coupled traceability for ATL, in: European Conference on Model Driven Architecture, Traceability Workshop ECMDA-TW, 2005, pp. 29–37.

[9] Y. Laghouaouta, A. Anwar, M. Nassar, B. Coulette, A graph based approach to trace models composition, J. Softw. JSW 9 (11) (2014a) 2813–2822, doi:10.4304/jsw.9.11.2813-2822.

[10] Y. Laghouaouta, M. Nassar, A. Anwar, J. Bruel, On the use of graph transformations for model composition traceability, in: IEEE 8th International Conference on Research Challenges in Information Science, RCIS 2014, 2014b, pp. 1–11, doi:10.1109/RCIS.2014.6861075.

[11] D.S. Kolovos, R.F. Paige, F. Polack, Merging models with the epsilon merging language (EML), in: 9th International ConferenceModel Driven Engineering Languages and Systems, MoDELS 2006, 2006, pp. 215–229, doi:10.1007/11880240_16.

[12] F. Jouault, I. Kurtev, Transforming models with ATL, in: Satellite Events at the MoDELS 2005 Conference, 2005, pp. 128–138, doi:10.1007/11663430_14.

[13] L. Chung, B.A. Nixon, E. Yu, J. Mylopoulos, Non-Functional Requirements in Software Engineering, Springer US, 2000, doi:10.1007/978-1-4615-5269-7.

[14] B. Wei, Z. Jin, L. Liu, A formalism for extending the NFR framework to support the composition of the goal trees, in: 17th Asia Pacific Software Engineering Conference, APSEC 2010, 2010, pp. 23–32, doi:10.1109/APSEC.2010.13.

[15] R. Laleau, A. Matoussi, A Survey of Non-Functional Requirements in Software Development Process, Technical Report TR-LACL-2008-7, LACL (Laboratory of Algorithms, Complexity and Logic), University of Paris-Est (Paris 12), 2008.

[16] S. Supakkul, L. Chung, A UML profile for goal-oriented and use case-driven representation of NFRs and FRs, in: Third ACIS International Conference on Software Engineering, Research, Management and Applications (SERA 2005), 2005, pp. 112–121, doi:10.1109/SERA.2005.19.

[17] D.S. Kolovos, R.F. Paige, F.A. Polack, Eclipse development tools for epsilon, Eclipse Summit Europe, Eclipse Modeling Symposium, 2006.

[18] C. Jeanneret, R. France, B. Baudry, A reference process for model composition, in: The 2008 AOSD Workshop on Aspect-Oriented Modeling, 2008, pp. 1–6.

[19] T.T.T. Nguyen, Codèle: Une Approche de Composition de Modèles Pour la Construction de Systèmes à Grande Échelle, Université Joseph-Fourier-Grenoble I, 2008 Ph.D. thesis.

[20] D. Kolovos, L. Rose, A. Garcia-Dominguez, R. Paige, The Epsilon Book, Eclipse, 2015.

[21] D.S. Kolovos, R.F. Paige, F.A. Polack, On-demand merging of traceability links with models, in: European Conference on Model Driven Architecture (ECMDA) workshop on traceability, 2006.

[22] R.F. Paige, N. Drivalos, D.S. Kolovos, K.J. Fernandes, C. Power, G.K. Olsen, S. Zschaler, Rigorous identification and encoding of trace-links in model-driven engineering, Softw. Syst. Model. 10 (4) (2011) 469–487, doi:10.1007/s10270-010-0158-8.

[23] N. Drivalos, R.F. Paige, K.J. Fernandes, D.S. Kolovos, Towards rigorously defined model-to-model traceability, in: European Conference on Model Driven Architecture, Traceability Workshop ECMDA-TW, 2008, pp. 17–26.

[24] W. Heaven, A. Finkelstein, UML profile to support requirements engineering with KAOS, IEE Proc. 151 (1) (2004) 10–28, doi:10.1049/ip-sen:20040297.

[25] O. Gotel, J. Cleland-Huang, J.H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, J.I. Maletic, The grand challenge of traceability (v1.0), in: Software and Systems Traceability, 2012, pp. 343–409, doi:10.1007/978-1-4471-2239-5_16.

[26] B. Amar, H. Leblanc, B. Coulette, C. Nebut, Using aspect-oriented programming to trace imperative transformations, in: the 14th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2010, 2010, pp. 143–152, doi:10.1109/EDOC.2010.12.

[27] B. Grammel, S. Kastenholz, A generic traceability framework for facet-based traceability data extraction in model-driven software development, in: the 6th ECMFA Traceability Workshop, ECMFA-TW 2010, 2010, pp. 7–14, doi:10.1145/1814392.1814394.

[28] J.M. Vara, V.A. Bollati, Á. Jiménez, E. Marcos, Dealing with traceability in the MDDof model transformations, IEEE Trans. Softw. Eng. 40 (6) (2014) 555–583, doi:10.1109/TSE.2014.2316132.

[29] R. Filman, T. Elrad, S. Clarke, M. Akşit, Aspect-oriented Software Development, first ed., Addison-Wesley Professional, 2004.

[30] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W.G. Griswold, An overview of aspectj, in: 15th European Conference Object-Oriented Programming ECOOP 2001, 2001, pp. 327–353, doi:10.1007/3-540-45337-7_18.

[31] O. Spinczyk, A. Gal, W. Schröder-Preikschat, Aspectc++: an aspect-oriented extension to the C++ programming language, in: The Fortieth International Conference on Tools Pacific: Objects for Internet, Mobile and Embedded Applications, Australian Computer Society, 2002, pp. 53–60.

[32] R.B. France, I. Ray, G. Georg, S. Ghosh, Aspect-oriented approach to early design modelling, IEE Proc. 151 (4) (2004) 173–186, doi:10.1049/ip-sen:20040920.

[33] G. Rozenberg (Ed.), Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations, World Scientific, 1997.

[34] M. Andries, G. Engels, A. Habel, B. Hoffmann, H. Kreowski, S. Kuske, D. Plump, A. Schürr, G. Taentzer, Graph transformation for specification and programming, Sci. Comput. Program. 34 (1) (1999) 1–54, doi:10.1016/S0167-6423(98)00023-9.
[35] G. Taentzer, K. Ehrig, E. Guerra, J. De Lara, L. Lengyel, T. Levendovszky, U. Prange, D. Varro, S. Varro-Gyapay, Model transformation by graph transformation: a comparative study, Workshop Model Transformation in Practice, 2005.
[36] G. Taentzer, AGG: A graph transformation environment for modeling and validation of software, in: Second International Workshop Applications of Graph Transformations with Industrial Relevance, AGTIVE 2003, 2003, pp. 446–453, doi:10.1007/978-3-540-25959-6_35.
[37] D. Varró, A. Pataricza, Generic and meta-transformations for model transformation engineering, in: UML 2004 - 7th International Conference on the Unified Modelling Language: Modelling Languages and Applications, 2004, pp. 290–304, doi:10.1007/978-3-540-30187-5_21.
[38] J. Whittle, P.K. Jayaraman, MATA: a tool for aspect-oriented modeling based on graph transformation, in: Models in Software Engineering, Workshops and Symposia at MoDELS 2007, 2007, pp. 16–27, doi:10.1007/978-3-540-69073-3_3.
[39] L.M. Garshol, BNF and EBNF: What are They and How do they Work?, (http://www.garshol.priv.no/download/text/bnf.html).
[40] T. Arendt, E. Biermann, S. Jurack, C. Krause, G. Taentzer, Henshin: advanced concepts and tools for in-place EMF model transformations, in: 13th International Conference on Model Driven Engineering Languages and Systems MODELS 2010, 2010, pp. 121–135, doi:10.1007/978-3-642-16145-2_9.
[41] E. Gansner, E. Koutsofios, S. North, Drawing Graphs with Dot, Technical Report, AT&T Research, 2006.
[42] E.R. Gansner, S.C. North, An open graph visualization system and its applications to software engineering, Softw. Pract. Exp. 30 (11) (2000) 1203–1233, doi:10.1002/1097-024X(200009)30:11⟨1203::AID-SPE338⟩3.0.CO;2-N.
[43] J. Bézivin, S. Bouzitouna, M.D.D. Fabro, M. Gervais, F. Jouault, D.S. Kolovos, I. Kurtev, R.F. Paige, A canonical scheme for model composition, in: Second European Conference on Model Driven Architecture - Foundations and Applications, ECMDA-FA 2006, 2006, pp. 346–360, doi:10.1007/11787044_26.
[44] Y. Laghouaouta, A. Anwar, M. Nassar, J. Bruel, A generic traceability framework for model composition operation, in: Enterprise, Business-Process and Information Systems Modeling - 16th International Conference, BPMDS 2015, 20th International Conference, EMMSAD 2015, Held at CAiSE 2015, Stockholm, Sweden, June 8–9, 2015, Proceedings, 2015, pp. 461–475, doi:10.1007/978-3-319-19237-6_29.
[45] A. Schürr, Specification of graph translators with triple graph grammars, in: Graph-Theoretic Concepts in Computer Science, 20th International Workshop, WG '94, Herrsching, Germany, June 16–18, 1994, Proceedings, 1994, pp. 151–163, doi:10.1007/3-540-59071-4_45.
[46] F. Klar, S. Rose, A. Schürr, Tie-a tool integration environment, in: Proceedings of the 5th ECMDA Traceability Workshop, 2009, pp. 39–48.
[47] E. Guerra, J. de Lara, D.S. Kolovos, R.F. Paige, Inter-modelling: from theory to practice, in: Model Driven Engineering Languages and Systems - 13th International Conference, MODELS 2010, Oslo, Norway, October 3–8, 2010, Proceedings, Part I, 2010, pp. 376–391, doi:10.1007/978-3-642-16145-2_26.
[48] S. Hildebrandt, L. Lambers, H. Giese, J. Rieke, J. Greenyer, W. Schäfer, M. Lauder, A. Anjorin, A. Schürr, A survey of triple graph grammar tools, Electr. Commun. EASST 57 (2013).