Applied Computer Systems doi: 10.1515/acss-2016-0006

2016/19

Enhancing Conflict Resolution Mechanism for Automatic Model Composition

Nisrine El Marzouki[1], Oksana Nikiforova[2], Younes Lakhrissi[3], Mohammed El Mohajir[4]

[1, 3, 4] *Sidi Mohamed Ben Abdellah University, Morocco,* [1, 2] *Riga Technical University, Latvia*

Abstract **– Despite the fact that model composition paradigm becomes very important and most commonly used, the support for their cooperation has not reached its full strength, especially in managing composition conflicts, because it's often divided between and confused with other model composition features. This makes handling and dealing with these conflicts a crucial activity in the composition process. Models need to be put under version control in order to manage the probable conflicts, facilitate collaboration and control change. Therefore, a solution for detecting and resolving conflicts is needed. In this paper, we present a composition conflict resolver presented in the form of a repository that helps manage composition conflicts and analyse the model and operations performed on it.**

Keywords **– Conflict Detection, conflict resolution, conflicting model composition, model composition**

I. Introduction

In general, incompabilities between independently developed modules are hard to solve. For simple conflict cases, the current technology provide relatively simple mechanisms. On the one hand they are easy to understand and to apply, but on the other hand they fail for even slightly more difficult problems. In this context, the composition of separate concerns is a cornstone for the construction of complex software, however a large number of composition concepts have evolved and been successfully put into practice, but their abilities to cope with composition conflicts are mostly limited, that why defining a repository to manage conflicts seems a major task in large model composition projects.

Indeed, in this paper we enhance the concept of managing composition conflict through a generic repository dedicated to the resolution of potential composition conflict in order to uncover composition errors and ensure that a composed model is produced with a level of credibility.

The main new idea of the research is to propose a general repository for model composition activities, which can be applied to conflicting model composition errors in particular. As the first step toward a global repository, our research will be based on a set of actions for detecting and correcting composition conflicts.

The remainder of this paper is structured as follows: Section 2 presents some common concepts and definitions because we recognise that without solid foundation it will not be possible to produce any new solution for managing model composition conflict. Sections 3 and 4 present a compact outline of our solution. A classification of conflicts detection and resolution strategies are given is Section 5. After a discussion of related studies in Section 6, we conclude the paper with a summary of our future research.

II. Background

There is little compromise in model composition jargon, and even less on the basic specifications of a model composition solution. To address this issue, the authors present in this section a common set of concepts and definitions for model composition in order to use this assessment to drive the key characteristics of our repository.

A. Concepts and Definitions

We propose a set of definitions for a model composition framework. They are an extraction of previous research [2] [3]. The formal definitions are intended as a starting point for a new solution. The approaches presented in [4] follow standards of model-driven architecture. This means they all have models as the central concept. The models are represented as graphs. In this case it is straightforward to converge to a graph model representation.

Definition 1 (Model). A model can be simply defined as a representation or abstraction of the system. By analogy with the world of programming, an executable program represents the modeled system while the source code of this program represents the model. From this analogy, we can use a new relationship called representedBy [1] between the model and the modeled system. In the MDA concept, the definition of a modeling language takes the form of a model, called metamodel [5]. The result is a new relationship called conformTo [1], which connects a model to the language in which it is expressed. (See Fig. 1).

Definition 2 (Metametamodel). A metametamodel is a model with a unique reference model. It defines the structure that must have any metamodel.

Definition 3 (Metamodel). A metamodel defines the structure that must have every model in accordance with this metamodel, it means that any model must follow the structure defined by its metamodel.

Definition 4 (Metamodeling). The purpose of metamodeling is to set a framework for defining modeling languages. Indeed, with the onset of MDA, space models is becoming broader and takes all dimensions of application (business, technical, level of abstraction, etc.) [6]. The idea is to use as many modeling languages as possible to express these models. These languages are dedicated to a particular domain (Domain Specific Modeling Languages – DSMLs). Metamodeling offers ways to define these DSMLs, study their relationship and thus control their complexity.

The approaches studied before [4] afford a way to harness the mapping between models in the context of model composition. In EML [8], the comparison rules are a central concept to produce a weaving model. First they specify merge rules that take as input the result of the comparison rules and then give as an output a set of relationships between the model elements presented in a form of weaving model. AMW [7] present the weaving model as a set of matching and merging links.

Definition 5 (Model transformation). A model transformation is defined by the operation of generating one or more target models from one or more source models.

Based on the Model driven Architecture principles, a transformation model defined by a Mt model is the process that transforms a Ma model (conforms to a MMa metamodel) to an Mb model (conforms to a MMb metamodel). In a more formal way, we can define this operation by the function Mb: MMb = Transf (Ma: MMa, Mt: MMt).

Depending on the abstraction level change caused by the transformation and the nature of the Meta source and target involved, several types of transformation are considered. Thus, there are the so-called endogenous transformations whose source and target models conform to the same metamodel and so-called exogenous transformations whose source and target metamodels are different.

Moreover, if we take into consideration the elements matched by a transformation process we can distinguish three types of transformation:

- Simple transformations (1 to 1) associate to any element of the source model at most one element of the target model.
- Multiple transformations (M to N) that take as input a set of elements in the source model and produce as output a set of elements (usually different) of the target model.
- The update transformations also called changes on site (add, delete, change properties, etc.). These transformations are characterized by the absence of the target model and therefore directly affect the source model. Restructuring transformations (refactoring) is a typical example of this type of transformation.

On the other hand, if we consider the level change of abstraction as a criterion for comparison, there are two types of transformation: vertical and horizontal. A vertical transformation causes a change in level of abstraction, it is the case of refining PIM to PSM in MDA approach, and a transformation is called horizontal if the models involved belong to the same level of abstraction.

Definition 6 (Meta-association). A meta-association allows to define a relationship between two metaclasses. In fact, a meta-Association defines the structure of links between meta-object instances metaclasses connected by the meta-Association

Definition 7 (Compose operation). The compose operation MAB = Compose (MA, MB, CAB) takes two models MA, MB and a correspondence model CAB between them as input and combines their elements into a new output model.

In the model composition approaches [4] there are some differences in the terminology to specify what a composition is. Apart from composition, the second most employed term is merge. However it is advisable to separate merge and composition. Composition is a more general operation. The semantic is specified in different operations by a set of rules, and it varies from case to case. Merge, however, is a special case of model composition. Merge has information preservation constraints, i.e., all the information from the input models should be present in the output models, and no duplicate information.

Definition 8 (Merge operation). The merge operation MAB = Merge (MA, MB, CAB) takes two models MA, MB and a correspondence model CAB between them as input, and returns a model MAB including all the information from MA and MB, without duplicate information. The correspondence model is created by the match operation. It specifies the elements that are going to be merged.

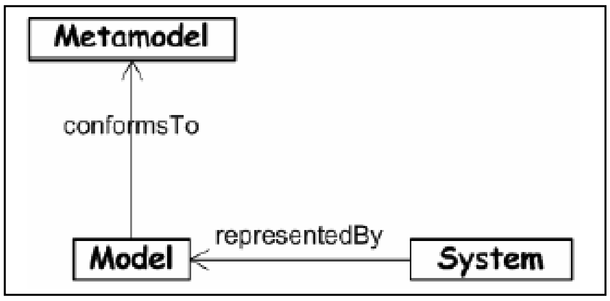

Fig. 1. Basics of model engineering [2].

B. Requirement for a Model Composition

The authors in their research [1] identified a core set of requirements for a model composition framework to complement the definitions for model composition presented in literature with a concrete set of minimal requirements for a model composition framework. Obviously, this is an initial set of requirements and it will likely need refinement after more practical experience and experiments with the frameworks have been carried out.

The authors in their research [1] identified a core set of requirements for a model composition framework to complement the definitions for model composition presented in literature with a concrete set of minimal requirements for a model composition framework. Obviously, this is an initial set of requirements and it will likely need refinement after more practical experience and experiments with the frameworks have been carried out.

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

- Means to reconstruct and reuse the composition elements;

In this category we find the AMW [8] approach that offers a weaving generic metamodel that defines the composition of links to a higher level of abstraction, the extension of this metamodel for defining the semantics of links depending on the application domain. The EML [7] supports generics by defining reusable generic rule libraries, and lets you merge models that conform to different Meta.

- Means to ensure consistency of models handled during the composition process.
- Means to define the ability of the approach to compose models without human intervention, in order to offer a more automated composition process.
- Means to determine the languages that support the traceability of changes and the possibility to keep track of each execution of a transformation rule.
- Means to identify corresponding elements in the models that are to be composed;
- Means to define how input elements can be matched in order to produce the target model;

Two desirable, practical requirements can be identified from [1]:

- A model composition framework should provide a means for formalizing the composition process by giving a formal definition of the composition operations and handled models;
- A model composition framework should be generic and independent of any modeling language to support future changes.

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

- A module for detecting and resolving composition conflict;
- A runtime module for model composition operations;
- A resolver for resolving failures and discrepancies that happen during the composition process;
- A refining mechanism for charging and preserving the models structure;
- Checking and merging rules to validate the model composition operations;

III. Conflict within Model Composition

In this section we motivate the need for detecting and resolving conflict in a model composition process and identify some of the instructions that can be followed to detect and resolve composition conflicts. We focus our attention to the class diagrams inconsistencies.

Several studies [10], [11] have demonstrated that managing conflicts and dependencies can be done by identifying possible conflicts and give them a solution or a transformation rule. Thus, in this phase of harmonization, the authors have identified three possible conflicts— Lexical conflicts, structural conflicts and semantic conflicts.

Lexical conflicts: this kind of conflict is caused by the confusion of identifying lexically equivalent concepts; multiple meanings and synonyms of classes introduced in the design of inputs models coming from different teams of designers make the mission more difficult. Indeed, a synonym dictionary is used in order to identify mappings among domain concepts that have an equal semantic value. The great benefit of using synonym dictionaries is to pave the way for the domain specialists to explicitly apply their domain expertise in the matching process.

Structural conflicts: Here we distinguish between 2 types of conflicts—those related to the type of association between two classes, and those related to the inheritance hierarchy. The authors intend using the TreeDiff implementation available in [12]. Our choice was based on its ability to identify structural similarities between trees in reasonable time. The result of the Tree Diff algorithm is the detection of concept equivalence groups. They are represented as subtrees of the enriched ontologies. Concepts that belong to such groups are compared in order to identify if lexically equivalent pairs can also be identified among the ancestors and descendants of the original pair.

Syntactic conflicts: includes class invariants, constraints, and operations specifications formalized by pre- and post-conditions expressed in OCL. As defined in [13], here we can use the Typographic Similarity, —where a syntactic property of a mode element defines its structure. The signature is a collection of values assigned to a subset of syntactic properties in a model elements metamodel class. If an instance of a Classis an abstract class then isAbstract = true for the class, otherwise the instance is a concrete class, isAbstract = false. The set of syntactic properties used to determine a profile elements signature called a signature type [14].

Moreover, understanding of the UML rules of class diagram allows for an efficient detection of conflicts. In general a conflict occurs when changes performed in parallel not only interfere with the modified element, but also with others. For example, semantic conflicts may occur when a developer modifies an element that depends on another element modified by other developers [15]. As semantic conflicts are more difficult to detect, they can generate false negative conflicts. Therefore, understanding of the rules related to the diagrams can also help reduce false negative conflicts.

IV. A Repository for Composition Conflict

In this section, descriptions of the repository are followed by illustrated examples. The set of resolution rules defined in this paper is not intended to be a complete set, but serves as a starting point for the eventual definition of a complete set of managing model composition conflict.

A. Overview:

There are several approaches to detect conflicts related to model composition. In this paper we reason about managing conflicts as a separated module presented in the form of a repository which communicate in real time with a composition framework or consistency analyser and can be used in every phase during the composition process as presented in Fig. 2. Some approaches pursue a more global way of reasoning by looking at the composition conflict as a secondary module included in the implementation of others modules. To illustrate this, we take another look at this paradigm by trying to determine all kinds of conflicts interacting at several levels (Class, relationship, operations). The authors are convinced that it would not be possible to detect conflict by looking only at the whole composition process: a separate context is needed to determine that the problem exists and then to propose a solution. Indeed, our primary goal is to define a generic module presented in the form of a repository for the composition phases (comparison, weaving, verification), to precisely describe, detect and resolve composition conflicts.

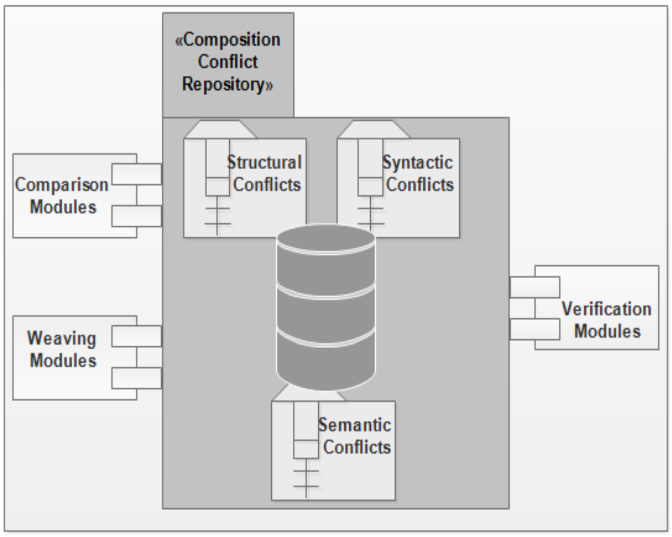

Fig. 2. A graphical concept of the composition conflict repository.

B. Composition Conflict Categories

The main goal of our repository is to model, detect and resolve composition conflicts related to the model composition process. Within this context, we identify several categories of composition conflicts. To analyse the causes of these conflicts precisely, we first identify probable causes. Next, we define explicit actions to resolve composition conflicts related to composition. We built a methodology based on UML rules (Table. I) that detects and analyses composition conflicts. As presented in the Fig. 2 we distinguish four categories of composition conflicts that can occur when a composition process occurs: syntactic conflicts, structural conflicts and semantic conflicts. The first type of conflict is the conflict caused by the multiple meanings and synonyms of models introduced as input models (classes, operations or attributes). The second type relates to structural conflicts between classes. Here we distinguish between those who are related to the type of association between two classes, those related to the inheritance hierarchy. The hierarchy of conflicts include inheritance cycles (which can appear when you want to merge the hierarchies from input models) and the level of conflict. The third type of conflict involves semantic conflicts that concern modeling elements. This includes class invariants, constraints, and operations specifications formalised by pre- and post-conditions expressed in OCL (Object Constraint Language). (See Table. II).

TABLE I

UML SEMANTICS RULES [26]

Number	**Rule**
1	In a mapping relationship such as Derivation, it is usually formal and unidirectional. In other cases, such as Trace, it is usually informal and bidirectional. The mapping expression is optional and may be omitted if the precise relationship between the elements is not specified.
2	In a generalization relationship, for each attribute in the superclass, an equivalent attribute is inferred for the subclass unless the attribute in question has a private visibility.
3	In a generalisation relationship, for each operation in the superclass, an equivalent operation is inferred for the subclass unless the operation in question has a private visibility.
4	In a generalisation relationship, for each association in the superclass, an equivalent association is inferred for the subclass.
5	In a realisation relationship, for each attribute of the interface, an equivalent attribute is inferred for the class.
6	In a realisation relationship, for each operation of the interface, an equivalent operation is inferred for the class.
7	If a class is abstract and all its methods are abstract then, if there is an interface that has the same name and the same elements, they are semantically equivalent.

TABLE II

THE DIFFERENT CONFLICT CATEGORIES TO DEAL WITH IN THE COMPOSITION PROCESS [26]

Conflict Category	**Conflict Types**	**Modeling element involved**
Syntactic	Synonymy	Class, operation, attribute
	Polysemy	
Structural	Inheritance cycle	Class
	Different level in hierarchy	Generalisation
	Different types of associations	Association
Semantic	Contradictory semantic assertions	Class, operation, attribute

V. Resolution Strategies

To resolve the conflicts presented in the previous part, we propose an interactive approach in three steps to address probable composition errors. The first step solves conflicts of polysemy and synonymy class hierarchy and conflicts. The second step can solve hierarchy conflicts using the same process. The third step solves conflicts over the semantic assertions.

A. Step 1: Treatment of Polysemy and Synonymy Models

In general, the classes of a system are supposed to have different names. But as the input models can be made by various teams' designers, the risk of having identical names for classes with different roles is strong. Hence, there is a need for treatment polysemy in order to achieve consistent models. The resolution strategy follows these instructions:

- Identify models that have the same name in the input models.
- Determine if they have the same responsibilities.
- Solve otherwise the conflict by renaming models.
- Choose new names for these models.

Check that these new names will not make a new problems of multiple meanings in the various dictionaries of classes; otherwise, we should choose other names until there are no more problems.

Figure 3 below summarises the key steps in the process of dealing with polysemy conflicts between input models.

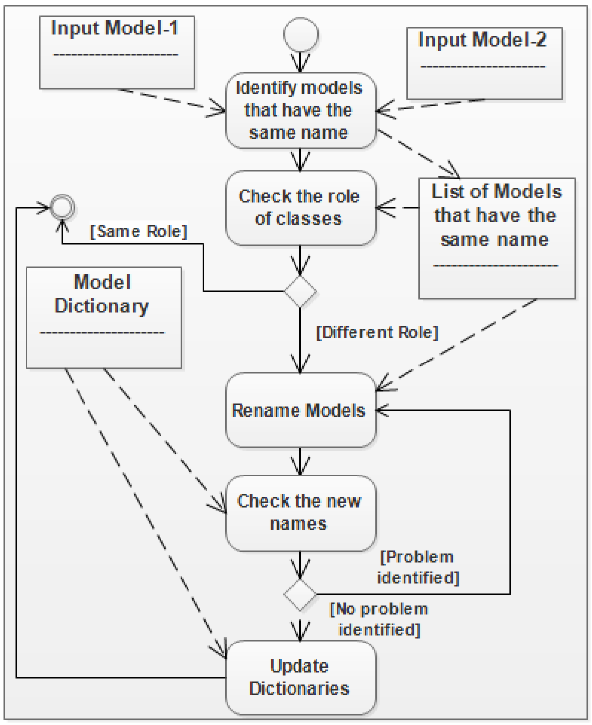

Fig. 3. A graphical concept of the composition conflict repository.

Treatment of synonymy is also an important step in the composition process to avoid duplication and ensure consistency between models. This treatment consists of:

- Identifying on the input models if there are any models that have the same role and different names;
- Renaming synonyms models that have the same name;
- Updating class dictionaries affected by this renaming.

B. Step 2: Treatment of Structural Inconsistency Conflicts

The resolution of this type of conflict aims to eliminate the structural inconsistencies between input models. This category includes conflicts of cycle' inheritance. These are problems of parent classes at different hierarchical levels, and inconsistencies between the types of association.

In case of inconsistency of the types of association between classes, we developed the following priorities based on the rules presented in Table. I and some previous research [27] [28]:

- Association vs Navigable Association: In this case the association predominates, because otherwise we risk losing information. Thereby a navigable association is bidirectional by default. This means that if an association exists between two classes, then both objects know about each other. If "A" is the source class and "B" is the target class, the arrowhead would be placed on the "B" side of the association. A navigable association of this type means that at runtime object "A" knows about object "B", but object "B" has no knowledge of or visibility into object "A".
- Navigable associations in opposite direction: The relationship between the two models is transformed into association (without navigation) that provides access to the source information.
- Association vs Composition: The relationship becomes an aggregation; thus, to guarantee the concept of compound.
- Association vs. Aggregation: Aggregation predominate because it conveys structural information that we should maintain.

C. Step 3: Treatment of Semantic Conflicts

To automate the processing of this type of conflict, we must call a "semantic" tool in order to analyse OCL expressions, detect conflicts and redundancies between semantic assertions established for an element defined in multiple input models. An example of resolving this kind of problem will be presented in the next section.

VI. Analysis of Conflicts Related to Composition

As we have previously identified different types of conflicts, in this section we discuss the various possibilities for handling and resolving every kind of conflict during the composition process.

A. Conflict Identification Example: University Artifact.

To show the different composition conflicts in a real case we will refer to a modeling system of the university artifacts (Fig. 3) made by different teams' designer, this example will illustrate the key elements of our solution.

We focus on the composition of two class diagrams, which correspond to different team's point of view.

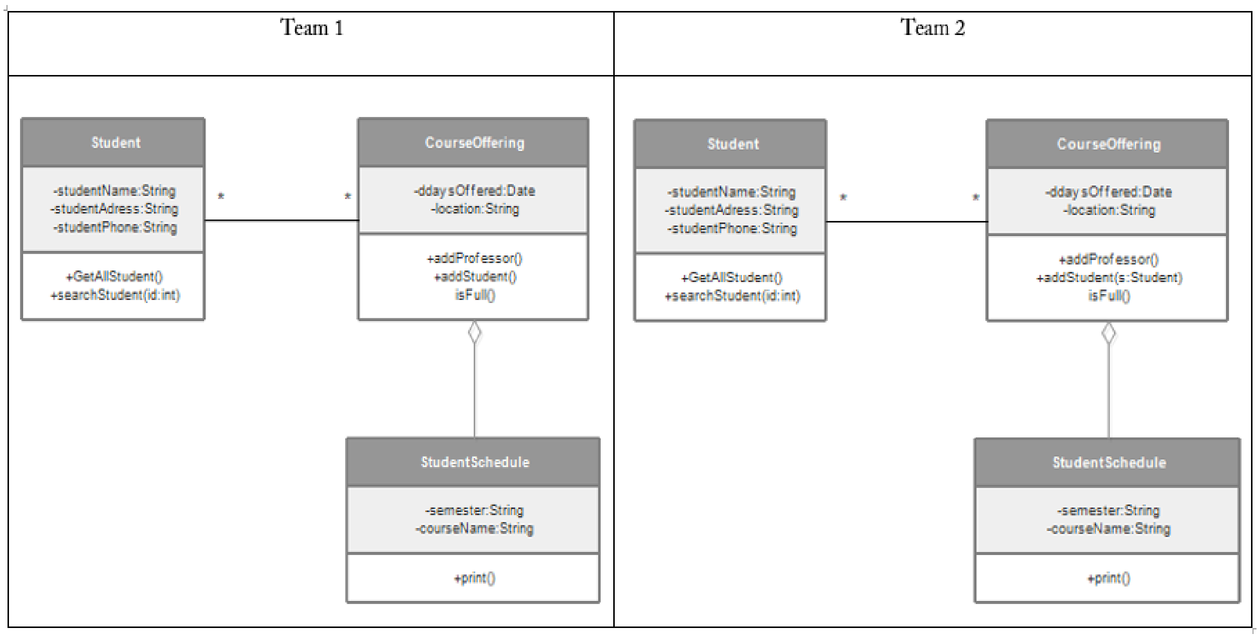

Fig. 4. Class diagrams made by two different teams: University artifacts.

Referring to Fig. 5, we consider operations addStudent() and addStudent(s:Student) in a class CourseOffering that is part of a class diagram for the university artifact, the operation addUser() defined by the first team in a class named CourseOffering adds a student to a collection of student. The addStudent operation in the second context of modelling calls the addStudent(s:Student) operation if and only if the user calling the operation is recognized to add a student. The addStudent(s:Student) operation adds a user to the list. This, the composition of these two different operations produces a conflict because the two operations have different framing and involved in different context. This is an example of a property conflict – a property conflict occurs when two matching elements (elements with the same name and syntactic type) are associated with conflicting properties. In this example, the intention is to merge the addStudent() operation made by a team with the addStudent(s: Student) operation produced by another team. To resolve this conflict, and based on some composition directives [8], we should rename the addStudent(s:Student) operation (First Team) to checkAndAddStudent, and keep the addStudent() operation in the same name. Renaming elements is not always a good solution to resolve this kind of conflict.

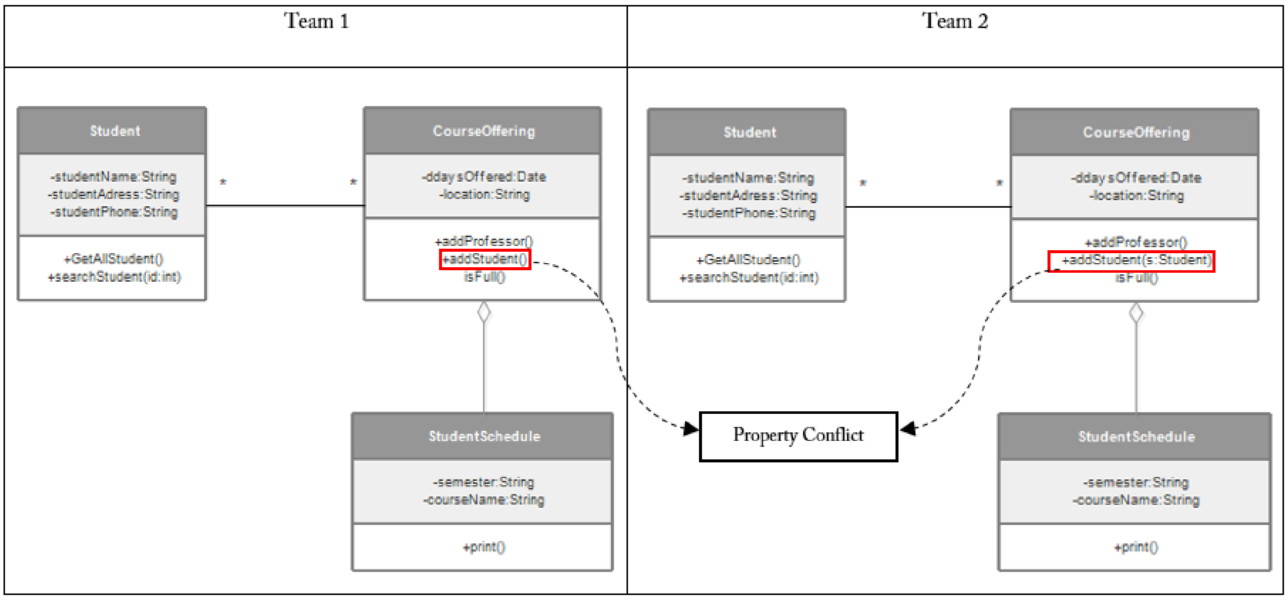

Fig. 5. An example of property conflict during the composition process of two class diagram.

As presented in Fig. 6, the first class diagram contains a class CourseCapacity with an attribute maxStudent: int, which has the constraint {maxStudent = 10}. The second class diagram with the same name (CourseCapacity) and the same attribute (maxStudent), but different constraint {maxStudent = 12}. Indeed, applying a merge operation to the two class diagrams will provide a property conflict, particularly in matching attributes because their constraint ({maxStudent = 10 and maxStudent = 2}) is incoherent. Yet, specifying an operation that will replace the other one can be a good directive to resolve this conflict. In this case, the properties operation with the high level of priority crushed the properties of the overloaded elements. This solution has also a poor side, because overriding relations can provide a cycle of conflict, especially when the two elements have the same level of priority and are in direct link with dominant properties. This to say that some time, it is necessary during the composition process to apply others operations (add, delete) to the elements in the goal to ensure producing a refined result. [6]

Our repository can be used to apply any kind of possible resolution during the composition process. If we take the example of an association, to guarantee and facilitate aces to an element, an association may be added or deleted (depending on the type of conflict) in order to avoid a security risk.

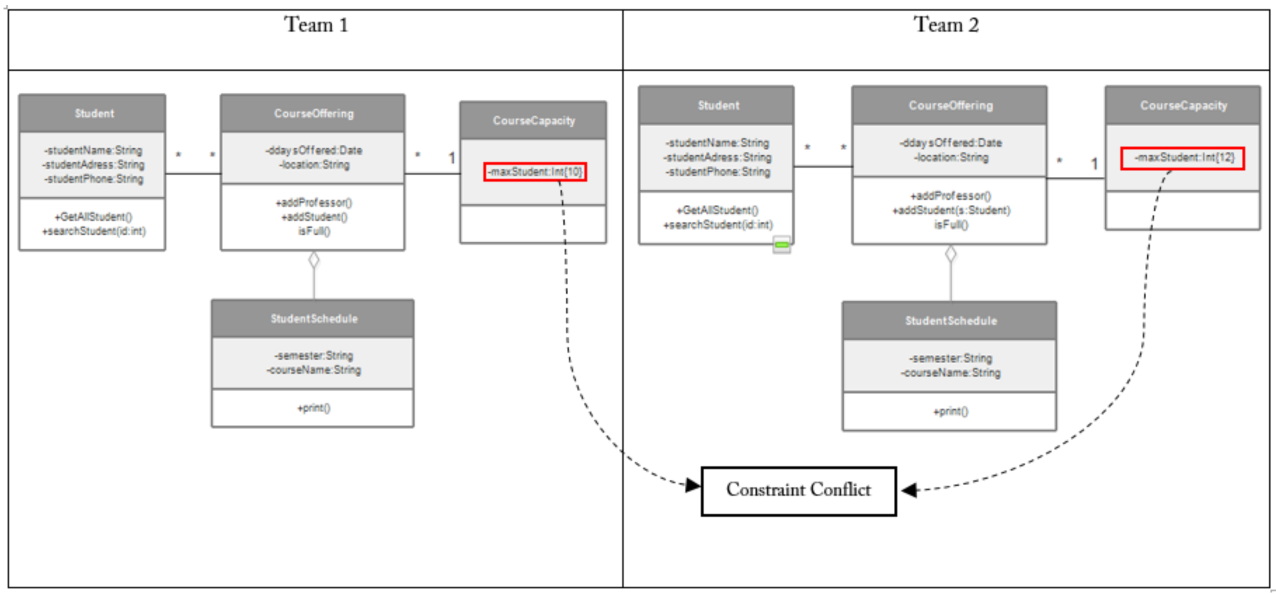

Fig. 6. An example of constraint conflict during the composition process of two class diagram.

B. Conflict Resolution

In this section we present many tables that summarise actions for resolving model composition conflict at several levels based on some previous research [25], [29].

TABLE III

MODEL COMPOSITION CONFLICT RESOLUTION AT CLASS LEVEL [29]

Operation	Description	Conflict Type
	Class-Level	
Reappoint Class	change the name of the class and update the dictionary renaming	Lexical and syntactical
Displace Class	Remove the current class and move all its properties to another class.	Lexical and syntactical
Take out Class	Move the property from the current class to a new one.	Lexical and syntactical

TABLE IV

MODEL COMPOSITION CONFLICT RESOLUTION AT RELATIONSHIP LEVEL [29]

	Relationship-Level	
Take out Hierarchy	During an inheritance hierarchy, we add a new class to a subclass.	Structural
Take out Subclass	Move the property from the current subclass to a new one.	Structural
Take out Superclass	Move the property from the current superclass to a new one.	Structural
Crash Hierarchy	Delete a class from an inheritance hierarchy.	Structural

TABLE V

MODEL COMPOSITION CONFLICT RESOLUTION AT METHOD LEVEL [29]

	Method-Level	
Step Down Method	Displace a method from a subclass, in order to put it in a class that require it more.	Structural
Step Up Method	Displace a method from a class to a superclass.	Structural
Rename Method	change the name of the method and update the dictionary renaming	Lexical and syntactical
Reappoint Method	change the name and the accessibility of the class and update the dictionary renaming	Syntactical
Raise Method	copy the same method in several classes	Syntactical
Displace Method	Remove the method from the current class and move all its features to another class.	Syntactical

TABLE VI

MODEL COMPOSITION CONFLICT RESOLUTION AT FIELD LEVEL [29]

Field-Level		
Step Down Field	Displace a field from a subclass, in order to put it in a class that require it more.	Structural
Step Up Field	Displace a field from a class to a superclass.	Structural
Displace Field	Remove the field from the current class and move all its features to another class.	Structural
Reappoint Field	change the name of the field and update the dictionary renaming	Lexical and syntactical
Decline Field	Remove the field from the current class and move all its features to another class.	Syntactical
Raise Field	copy the same field in several classes	Syntactic
Encapsulate Field	Increase accessibility to a field through the creation of getter and setter.	Syntactical

VII. RELATED WORK

In [19], the authors propose dividing the system into two types of interference in order to identify automatically classes of interference and allow developers to pick up potentially unwanted flow information. However, this is not presented as a framework for detecting conflict; thereby it focuses on the interactions in the weaving phase, while we focus on conflicts generated in any phase during the composition process.

In [20], Katz shows how to detect interactions that invalidate desirable properties, he proposes a new method of regression, checking and verification with a possible division into static analysis. In our research, we do not focus on checking undesirable system properties, and do not take in consideration that the programs can be augmented with specifications.

In [21], [22], [23], Clarke et al. describe a global model composition approach based on subjects, The compositions of subjects as a particular view of the comprehensive system include many operations that allow conflict resolution through the application of several measurement between conflicting elements, but nothing further. They describe directives that support the composition of constraints, and the deletion of model elements.

In [24], Douence et al. detect interactions between aspects using static analysis. When conflicts are detected, they can be resolved by extending the specification of the aspects. In this paper, we use graph transformations to simulate aspect compositions, thereby also modeling part of the semantics of the language. In our research we focus on how to detect and resolve conflicts related to the mode composition process.

VIII. CONCLUSION AND FUTURE WORK

This paper defines a set of composition conflict resolution presented in a form of repository that facilitates the customisation of model composition. This repository can form the basis for the development of tools to support the model composition process from the comparison phase to the verification phase. The repository also provides a common vocabulary for describing composition conflict actions. Illustrated examples demonstrate the use of each action.

The main challenge of the present research is to contribute to the understanding of composition conflicts, in particular within the scope of structural composition. To this extent we propose and illustrate a systematic approach to analyze such composition conflicts in a precise and concrete manner. We propose several actions to express conflict detection and resolution rules. These actions have been introduced to deliver a precise explanation why and when some forms of composition cause a conflict, and to ensure that the categories are not overlapping. Also, the precise formulation makes it possible to perform the conflict detection fully automatically, for example, as a separate module which communicate in real time with a composition framework or consistency analyser.

The actions defined by our repository are expressive in the sense that they can specify common composition conflict actions such as renaming, replacing models, and at the same time they can be used to specify creation and removal of model elements, making it possible to significantly alter how models are composed.

Empirical evaluation is needed to validate the repository in real world design settings. Specifically the amount of effort required to specify the kinds of resolution that are required in real world designs needs to be empirically evaluated; the development of a tractable method of identifying conflicts in the composed model needs to be investigated; and the currently defined repository needs to be evaluated for its ability to support the kinds of composition conflict that actually occur. This evaluation could result in the specification of some common detection conflict strategies to manage the complexity of specifying compositions and could be an area of future research. We are also exploring how to express the applicability and consequences of using a generic repository in terms that it can interact in any step of the composition process. We plan to investigate the use of the Object Constraint Language [25] for this purpose.

ACKNOWLEDGEMENTS

The research presented in this article has been supported by the research grant of the Erasmus+ EU program for a mobility period between *Sidi Mohamed Ben Abdellah University* and *Riga Technical University* and also by Grant of Latvian Council of Science No. 09.1269 "Methods and Models Based on Distributed Artificial Intelligence and Web Technologies for Development of Intelligent Applied Software and Computer System Architecture".

REFERENCES

[1] J. Bézivin, S. Bouzitouna, M. D. Del Fabro, M. Gervais, F. Jouault, D. Kolovos, I. Kurtev, R. Paige, “A Canonical Scheme for Model Composition,” *Proc. of ECMDA-FA*, LNCS 4066, Springer-Verlag, 2006, pp. 346–360. http://dx.doi.org/10.1007/11787044_26

[2] J. Bézivin, F. Jouault, D. Touzet, “An introduction to the ATLAS Model Management Architecture,” LINA, Nantes University, Feb. 2005.

[3] B. Baudry, “An overview of approaches Oriented Models Aspects,” MOA days (Models oriented aspects), Toulouse, 2009.
[4] N. E. Marzouki, Y. Lakhrissi, M. Elmohajir, “A Study of Behavioral and Structural Composition Methods and Techniques,” 978-1-4673-7689-1/IEEE-16-April 2016.
[5] B. Combemale, “Meta modeling approach for simulation and model checking – Application to process engineering,” Thesis of the National Polytechnic Institute of Toulouse, July 2008.
[6] A. Muller, “Construction of systems by applying parameterized models,” Thesis of the University of Lille 1, June 2006.
[7] Atlas Model Weaver Project Web Page. (2005). Available: http://www.eclipse.org/gmt/amw/
[8] EMF Eclipse. Eclipse modeling framework. (2006). Available: http://www.sysml.org/docs/specs/OMGSysML-v1.1-08-11-01.pdf
[9] R. Pottinger and P. Bernstein, “Towards Model Composition,” in *Proc. VLDB*, ACM, 2003.
[10] P. Kelsen and Q. Ma, “A Modular Model Composition Technique,” in D.S. Rosenblum and G. Taentzer (Eds.). *Fundamental Approaches to Software Engineering*, LNCS 6013, pp. 173–187, 2010, Springer-Verlag Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-642-12029-9_13
[11] I. H. Moghadam and M. O Cinneide, “Resolving Conict and Dependency in Refactoring to a Desired Design,” *e-Informatica Software Engineering Journal*, vol. 9, Issue 1, 2015. http://dx.doi.org/10.5277/e-Inf150103
[12] Y. Lin, J. Gray, and F. Jouault, “DSMDiff: a differentiation tool for domain-specific models,” *European J. of Information Systems*, vol. 16, pp. 349–361, 2007. http://dx.doi.org/10.1057/palgrave.ejis.3000685
[13] M. Alanen and I. Porres, “Difference and Union of Models,” in J. Whittle, and G. Booch (eds.), *“UML” 2003 - The Unified Modeling Language. Modeling Languages and Applications* Stevens, 2003. http://dx.doi.org/10.1007/978-3-540-45221-8_2
[14] Y. R. Reddy, S. Ghosh, R. B. France, G. Straw, *et al.*, “Directives for Composing Aspect-Oriented Design Class Models,” *Transactions of Aspect-Oriented Software Development I*, vol. 3880, LNCS, pp. 75–105, Springer. http://dx.doi.org/10.1007/11687061_3
[15] C. Brun and A. Pierantonio, “Model differences in the eclipse modelling framework,” *The European J for the Informatics Professional*, UPGRADE, vol. IX, issue 2, pp. 29–34, 2008.
[16] H. Zhang, “Delay-insensitive networks,” M.S. thesis, University of Waterloo, Waterloo, ON, Canada, 1997.
[17] A. Rezi and M. Allam, “Techniques in array processing by means of transformations,” *Control and Dynamic Systems, Multidemsional Systems: Signal Processing and Modeling Techniques*, C. T. Leondes, Ed. San Diego: Academic Press, vol. 69, pp. 133–180, 1995. http://dx.doi.org/10.1016/S0090-5267(05)80040-4
[18] N. Osifchin and G. Vau, “Power considerations for the modernization of telecommunications in Central and Eastern European and former Soviet Union (CEE/FSU) countries,” in *Second Int. Telecommun. Energy Special Conf.*, 1997, pp. 9–16. http://dx.doi.org/10.1109/TELESC.1997.655690
[19] M. Rinard, A. Salcianu, and S. Bugrara, “A classification system and analysis for interactions in aspect-oriented programs,” in *Foundations of Software Engineering (FSE)*, pp. 147–158. ACM, Oct. 2004. http://dx.doi.org/10.1145/1041685.1029917
[20] S. Katz, “Diagnosis of harmful aspects using regression verification,” in C. Clifton, R. Lammel, and G. T. Leavens (eds.), *FOAL: Foundations Of Aspect-Oriented Languages*, pp. 1–6, Mar. 2004.
[21] S. Clarke and J. Murphy, “Developing a tool to support the application of aspect-oriented programming principles to the design phase,” in *Proc. of the Int. Conf. on Software Engineering*, ICSE '98, Kyoto, Japan, April 1998.
[22] S. Clarke, W. Harrison, H. Ossher, and P. Tarr. “Separating concerns throughout the development lifecycle,” in *Proc. of the 3rd ECOOP Aspect-Oriented Programming Workshop*, Lisbon, Portugal, June 1999.
[23] S. Clarke. “Extending standard UML with model composition semantics,” *Science of Computer Programming*, vol. 44, issue 1, pp. 71–100, Elsevier Science, July 2002. http://dx.doi.org/10.1016/S0167-6423(02)00030-8
[24] R. Douence, P. Fradet, and M. Sudholt, “Composition, reuse and interaction analysis of stateful aspects,” in K. Lieberherr (ed.), *Proc. 3rd Int. Conf. on Aspect-Oriented Software Development,* AOSD ‘2004, pp. 141–150, ACM Press, Mar. 2004. http://dx.doi.org/10.1145/976270.976288
[25] The Object Management Group (OMG). *Unified Modeling Language*. OMG, Available: http://www.omg.org/docs/formal/03-03-01.pdf. Version 1.5, March 2003.
[26] UML Syntax and Semantics Guide V1.1, OMG, ad/97-08-03, Available: www.rational.com/UML.
[27] A. Anwar, “Formalization by IDM approach to compose models in the VUML profile,” Thesis, Toulouse University, Dec. 2009.
[28] A. Kriouile, “BOOM, an object-oriented method of analysis and design through views,” Thesis, Faculty of Sciences Rabat, Morocco, 1995.
[29] I. H. Moghadam and M. O Cinneide, “Resolving Conflict and Dependency in Refactoring to a Desired Design,” *e-Informatica Software Engineering J.*, vol. 9, issue 1, 2015, pp. 37–56. http://dx.doi.org/10.5277/e-Inf150103

Nisrine El Marzouki is a second-year PhD student at the Laboratory of Computer Science, Modelling and Systems of the Faculty of Science at Sidi Mohamed Ben Abdellah University, Morocco. His current research interest is model composition in the framework of model driven architecture approach.
E-mail: elmarzoukinisrinegmail.com

Oksana Nikiforova received the Doctoral degree in Information Technologies (system analysis, modelling and design) from Riga Technical University, Latvia, in 2001. She is a Professor at Riga Technical University. Her current research interests include object-oriented system analysis and modelling, especially the issues on model driven software development
E-mail: oksana.nikiforova@rtu.lv

Younes Lakhrissi received the doctoral degree in information technologies (system analysis, modeling and design) from Toulouse le Mirail University, France, in 2010. He is presently a Professor with the Sidi Mohamed Ben Abdellah University. His current research interests include object-oriented system analysis and modelling, especially the issues on Model Driven Engineering.
E-mail: younes.lakhrissi@gmail.com

Mohammed El Mohajir obtained his PhD in science from the University Catholic of Louvain (Belgium) in 1997. He was Doctor Research Associate from 1997 to 1998 at UCL, Belgium. He then moved to the United Kingdom to hold a Senior Research Associate position at the School of Mathematics at the University of East Anglia in Norwich. He joined Sidi Mohamed Ben Abdelah University in 2004. He is now full Professor at the department of computer sciences at the Faculty of Science Dhar Mahraz. His research interest is towards conceptual modeling, requirement engineering and entreprise Organizational modeling, decision-support Information Systems and Big Data Analytics.
E-mail: m.elmohajir@ieee.ma