A Process for Maintaining Heterogeneous Models Consistency through Change Synchronization

Mahmoud El Hamlaoui, Sophie Ebersold, Adil Anwar, Mahmoud Nassar, Bernard Coulette

A Process for Maintaining Heterogeneous Models Consistency through Change Synchronization

Mahmoud EL HAMLAOUI[1,2], Sophie EBERSOLD[1], Adil ANWAR[3], Mahmoud NASSAR[2] and Bernard COULETTE[1]
[1] University Toulouse 2 –Le Mirail
IRIT Laboratory, MACAO Team
5 Allée Antonio Machado, 31058 Toulouse, France
{mahmoud.el- hamlaoui, sophie.ebersold, bernard.coulette}@irit.fr
TEL: (+33)5 61 50 38 96- Fax: (+33) 5 61 50 41 73
[2] University of Med V Souissi
ENSIAS, SIME Laboratory, IMS Team
BP 713, Agdal Rabat, Morocco
nassar@ensias.ma
TEL: (+212)5-37 77 85 79- Fax: (+212)5 37 77 72 30
[3] University of Med V Agdal
EMI, Siweb Laboratory
BP 765, Agdal Rabat, Morocco
anwar@emi.ac.ma

***Abstract*—This paper falls into the context of modeling complex systems according to various viewpoints. More precisely, it presents an iterative process of heterogeneous models consistency management – by taking into account various types of evolution - based on building a correspondence model. In the case of models evolution, this process is intended to capture changes in the models, to list modifications to be made in the impacted models and finally to update the correspondence model for a future iteration.**

Keywords- evolution; changes; correspondence model; virtualization; impacts;

I. INTRODUCTION

Today, development of complex systems is based on a varied set of languages, tools and environments that are generally used separately by modeling experts working on different dimensions of a project.

Most of these complex systems are designed so that their multiple views are defined in different heterogeneous DSMLs (domain-specific modeling languages), representing projections of the global view of the system according to specific needs. In the avionics domain for example, it is common to develop various models corresponding to different viewpoints on a given system: mechanical, thermal, electrical, computing, etc.

To tackle consistency problems between heterogeneous partials models (views), it is commonly admitted that a "matching" phase is necessary. It is a way to connect those models via a set of relationships defined between them. We have investigated this issue by defining a correspondence meta-model that will be recalled in subsequent sections.

The question that arises then is "how to manage partial models evolutions?" Indeed, during the modeling or the maintenance phase of a complex system, designers working with specific DSLs according to their viewpoints tend to change the models on which they operate. This may cause inconsistencies since models are related so that the change of one of them may cause the inconsistency of the whole system. In fact, there is a need to reflect and adapt the change, or at least to identify the models that are impacted by it. To solve this issue we introduce a semi-automatic iterative process based on a proposed correspondence model, oriented to deal with evolution aspects.

In this paper, we focus on maintaining consistency by impacting changes as a result of partial models evolution.

The remainder of this paper is structured as follows. Section II, investigates related works. Section III, introduces the proposed correspondence model. Section IV, presents our process called "evolution process" and its different phases and, finally, the paper is concluded in Section V.

II. RELATED WORK

Several approaches in the literature treat one or several aspects of model evolution issue. In this section we focus on some of these approaches, namely COPE [7], EMFMigrate [5] and the one developed by Cicchetti et al. [3].

To lead this study, we have identified the following criteria: Heterogeneity, number of input artifacts and their types, mechanism of change detection, the adopted support of classification and the evolution level. These criteria − that should ideally be present in every approach − are defined below:

- Heterogeneity: expresses if the approach in question takes into account heterogeneous artifacts. As a reminder, we consider that two artifacts are heterogeneous if their modeling languages are themselves heterogeneous,
- Change detection: assesses how an approach proceeds to detect the elements of artifacts that have undergone an alteration,
- Number of input artifacts: since we are interested in multi-environment modeling, this criterion characterizes the possible limitation on the number of input artifacts,
- Types of artifacts: identifies the shape of representing artifacts. The latters are not necessarily models, they might be rules of transformation or other types of artifacts,
- Classification support: indicates whether the approach supports a classification of changes in order to assign to each kind of change a particular action. This is generally done in some phases preceding the evolution phase. It is interesting to take this criterion into account, because the classification of changes allows the automation of the whole evolution process or at least a part of it,
- Evolution level: characterizes the type of level: vertical or horizontal. Co-evolution, for example, is a vertical evolution level as its objective is to maintain the conformity relationship between a model and it's meta-model. The horizontal evolution level concerns changes at the same level between models, also called model migration.

TABLE I presents a synthesis of the studied approaches, based on these criteria. By analyzing it we can deduce that the evolution process has not yet reached maturity level. Firstly, studied approaches take into consideration only homogeneous models (i.e. derived from the same meta-model). Yet it is essential to be able to take into account heterogeneous models. Secondly, they do not define any classification support, a factor that we consider as mandatory in order to automatically manage changes and their impacts on models, through predefined actions. Last but not least, most of the approaches discussed above (except Cicchetti et al.), focus on the migration of models as a result of adaptation of their corresponding meta-models (co-evolution) to preserve the conformity relationship. That is to say that these approaches only treat the vertical level of evolution. Yet it is on the horizontal level that models synchronization is based

TABLE I. COMPARISON OF MODEL EVOLUTION APPROACHES

Criteria / Approaches	H	NA	TA	CD	CS	EL
Cope (Edapt)	No	2	M[1]	SA[3]	No	V[4]
EMFMigrate	No	2	M/T[2]	Manual	No	V
Cichetti & al.	No	2	M	Manual	No	H[5]

[1]: Model, [2]: Transformation rules, [3]: Semi-automatic, [4]: Vertical, [5]: Horizontal

To sum up, the identified approaches do not fully address these important aspects of system evolution. Also, different criteria are not considered, which limit their use to specific case studies.

III. CORRESPONDENCE META-MODEL

It was mentioned above that view-based models of the system are connected through a set of relationships. In other words the global system view is a couple {Vn, Rn} such that Vn is the set of views and Rn is a set of relationships instantiated from a correspondence meta-model. Therefore, before tackling the part relating to model evolution, we briefly describe a preliminary phase of the evolution process that is called "Matching". For this phase, we have defined a correspondence meta-model called "CMM" (see Fig 1). It has been designed to meet two main quality criteria: genericity and extensibility. CMM provides a "generic" part – common to all domains - that defines a syntactic description of most common types of correspondence. CMM can be extended depending on the specificities of the application domain under consideration, in order to support the concepts relating to specific business areas. A description of the concepts of the proposed CMM will be detailed in [6].

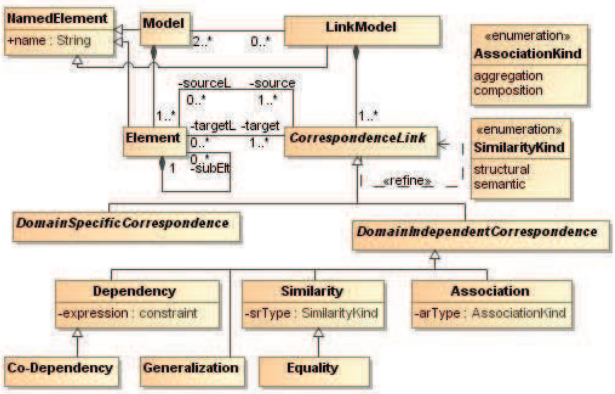

Fig 1. Overview of the correspondence meta-model (*CMM*)

Another property of the MMC is that its instance, the correspondence model (CM), is built in a virtual manner. That means that the CM does not contain any concrete data. A virtual model provides tools/users the illusion of working with a regular model whereas, in fact, all model manipulation requests are transparently redirected to elements contained in the virtualized contributing models [4]. In our case this means that CM only contains – physically– the relationships without the related elements which are located into the source models. During the matching phase, a virtual link is established between the instance of "Element" in the CM and the elements that still exist in the input models.

IV. EVOLUTION PROCESS

In this section, we present the "evolution process" which takes place after the matching phase mentioned in the introduction. This suggested process, represented in SPEM [8] (Fig 2), aims at describing the phases to perform after an

evolution of connected models (called input models), in order to maintain the consistency of the system. It involves two actors, namely, a domain expert who can be seen as an orchestrator of the system, and designers who are responsible for input models.

The suggested process consists of three major phases which are: change detection, change classification and change processing. The process takes as input the various models that may have evolved, and the correspondence model (CM). This latter is conform to the correspondence meta-model (CMM) obtained in the matching phase (out of scope of the evolution process). Firstly, a change detection step is triggered in order to trace changes that might have occurred on the various input models. These changes are specified (added) in an extended part of the CMM. Secondly a classification phase is performed. It aims, by involving the domain expert, at classifying the previous changes stored into CMM by producing a change list. This is performed in order to better manage impacts by assigning to each case a specific action. The final phase, called change processing, aims to migrate models by applying specific treatments to them. Some of them require the approbation of domain expert and designers.

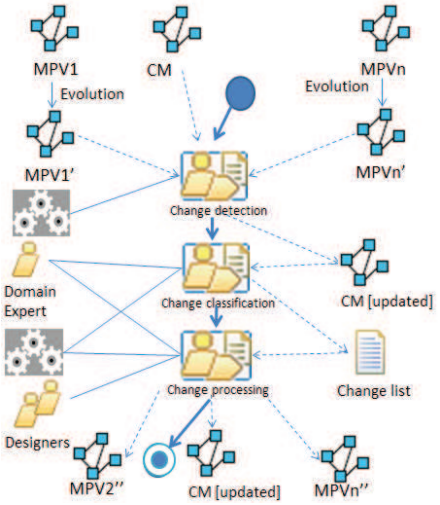

Fig 2. The whole evolution process (*for one iteration*)

A. Change detection

A change considered meaningful for one purpose may be irrelevant for another [9]. Therefore, we aim to provide three ways for triggering this phase, exploitable depending on the needs:

- On a model element change: Every change of model element causes the initialization of the phase,
- Periodical: The actors work independently on their respective models, and the change detection is triggered at specific times in the development cycle to evolve the elements in question,
- On expert user request: A contextual menu is implemented so that the actor could evaluate the need of triggering the phase.

1) Extension of CMM supporting model evolution

The *change detection* phase aims to detect the models elements that have undergone a change, i.e. elements that have been altered, deleted or added. Unlike the correspondence process that highlights the similarities and dependencies between (meta-) models elements, the result of this phase is the specification of discrepancies (deltas) caused by the evolution of one or several models elements. Based on these deltas, we will subsequently identify the model elements affected by the change and the necessary amendments to ensure the system consistency.
To describe these evolutions, we extend the CMM meta-model (see Fig 1) by adding a set of concepts, mimicking a CRUD [1] operation set.

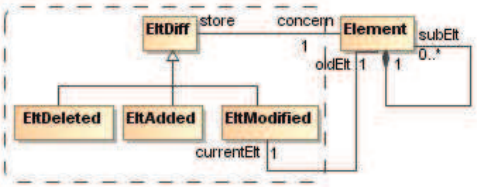

Fig 3. Extract of correspondence meta-model, oriented towards model evolution

As described in the Fig 3, several types of changes are taken into accounts which are represented by the following concepts:

- EltDiff: Abstract meta-class that stores through it specialization concepts, a trace of the changed elements,
- EltDeleted: Elements of models that no longer exist, as a result of a delete operation,
- EltAdded: New model elements that are added to the initial models,
- EltModified: New state of a model element that is defined as a result of an amendment to existing ones.

2) Enrichment of the correspondence model

The extension of the meta-model presented in the previous sub-section, define only where to store the different changes without defining the how part that will be the purpose of this sub-section. In order to supply the correspondence model (CM) with the different types of change, we exploit the comparison engine EMFCompare[2]. EMFCompare is a Framework that provides a generic algorithm for calculating differences between two versions of a model, based on distance calculating techniques. The provided result could be used in different ways. In our case, we used it as input for enriching the correspondence model, by filling in the elements have been changed

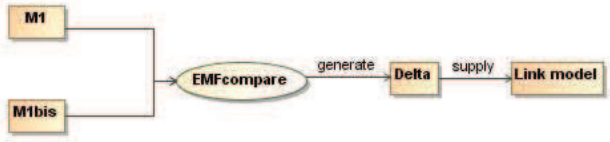

Fig 4. Identifying changes using EMFCompare

B. *Classification of changes*

The classification of changes is used to manage impacts by assigning to each type of change a special treatment. Therefore, we propose to classify them into two categories:

- "Automatic Evolution Category": contains changes that lead to automatic actions performed on models. For example, if we delete a model element, the relationship becomes orphan. We define an orphan relationship as a relationship for which one of its extremities (that are model elements) is missing. When a relationship is orphan it must be deleted from the correspondence model,
- "Monitored Evolution Category": includes actions that require a human assistance to decide about certain types of changes. For example, if one of the relationship-ends has been modified, it is the expert's responsibility to decide whether to maintain the relationship with the new ends or to modify one of them, if it still needs to exist.

We must note that, for each type of change, it is possible through the correspondence model to find for a specific element, the type of links and elements of models on the relationship extremities.

C. *Change processing*

To maintain the consistency of the system with regard to established relationships, model migration must be performed. In this phase, models are amended to take into account the identified changes and the modifications deemed by the experts to be impacted. On the one hand, the evolutions classified under "Automatic Evolution Category" will be handled automatically. On the other hand, evolutions classified under "Monitored Evolution Category" will result in a semi-automatic migration operation that offers evolution suggestions to guide the expert and help him to evolve the model elements of the system.

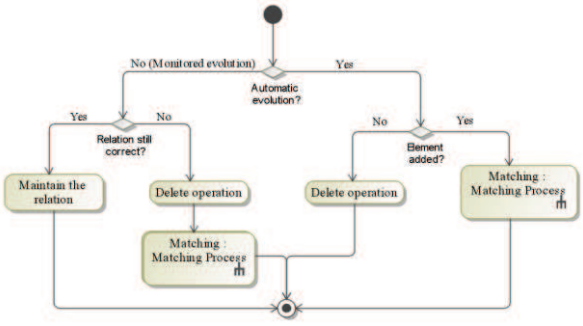

Fig 5. Detailed vision of the change impact activity

The figure above describes the process followed for the change processing phase. There are four global activities related to addition, modification, deletion and the maintaining of the relationship and two operations: matching and delete. The modification activity is a successive execution of deletion and matching.

V. CONCLUSION AND PERSPECTIVES

In the context of complex systems development, a set of heterogeneous and evolving view-based models must be managed. This brings out the need for a change management mechanism enabling impact of changes on the elements concerned and thereby ensuring the coherence of the system. In this paper we have addressed some maintenance issue in case of input models evolution. This is done through a semi-automatic process that uses an extended correspondence model with virtual access to the evolved element, allowing to (i) detect changes made in a given input model, (ii) handle the modifications according to a performed classification and (iii) update the correspondence model to maintain the consistency of the system. In a multi-environment modelling, several modifications can be performed simultaneously on different models. A perspective given to this paper is to coordinate and schedule the synchronization of data between model elements. In addition, how the integration of the proposed approach in a tooling suite can be achieved, is still to be investigated.

ACKNOWLEDGMENT

This paper describes the results of a research work in the scope of the PHC Volubilis MA/11/254

REFERENCES

[1] Franck Barbier, Sylvain Eveillard, Kamal Youbi, and Eric Cariou. Model-driven reverse engineering of cobol-based applications. *Information Systems Transformation. Architecture Driven Modernization Case Studies, Morgan Kauffman, Burlington, MA*, pages 283–299, 2010.

[2] C. Brun and A. Pierantonio. Model differences in the eclipse modeling framework. *UPGRADE, The European Journal for the Informatics Professional*, 9(2):29–34, 2008.

[3] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. Automating co-evolution in model-driven engineering. In *Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE*, pages 222–231. IEEE, 2008.

[4] Cauê Clasen, Frédéric Jouault, and Jordi Cabot. Virtualemf: a model virtualization tool. *Advances in Conceptual Modeling. Recent Developments and New Directions*, pages 332–335, 2011.

[5] D. Di Ruscio, L. Iovino, and A. Pierantonio. What is needed for managing co-evolution in mde? In *Proceedings of the 2nd International Workshop on Model Comparison in Practice*, pages 30–38. ACM, 2011.

[6] Mahmoud El Hamlaoui, Sophie Ebersold, Bernard Coulette, Adil Anwar, and Mahmoud Nassar. A process for defining a unique correspondence model to relate heterogeneous models. In *International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), Angers - France, 04/07/2013-06/07/2013*. SciTePress, 2013.

[7] M. Herrmannsdoerfer, S. Benz, E. Juergens, et al. Cope: A language for the coupled evolution of metamodels and models. In *1st International Workshop on Model Co-Evolution and Consistency Management*, 2008.

[8] OMG SPEM. Omg spem-v2.0. http://www.omg.org/spec/SPEM/2.0/PDF, April 2008.

[9] Y. Yu, T.T. Tun, and B. Nuseibeh. Specifying and detecting meaningful changes in programs. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 273–282. IEEE Computer Society, 2011.