A MDE Approach for Heterogeneous Models Consistency

Mahmoud El Hamlaoui[1], Saloua Bennani[1,2], Mahmoud Nassar[1], Sophie Ebersold[2] and Bernard Coulette[2]
[1]*IMS Team, ADMIR Laboratory, ENSIAS, Rabat IT Center, Mohammed V University in Rabat, Morocco*
[2]*SMART Team, IRIT Laboratory, University of Toulouse-Jean Jaurès, France*

Keywords: Process, Metamodel, Heterogeneous Models, Matching, Consistency, Correspondences, Impacts.

Abstract: To design a complex system, we often proceed via separation of viewpoints. Each viewpoint is described by a model that represents a domain expertise. Those partial models are generally heterogeneous (i.e conform to different metamodels) and thus performed by different designers. We proposed a matching process that links partial models through a virtual global model in order to create a complete view of the system. As models evolve, we should consider the impact of changing an element involved in a correspondence on other models to keep the coherence of the global view. So, we have defined a process that automatically identify changes, classify them and treat their potential repercussions on elements of other partial models in order to maintain the global model consistency.

1 INTRODUCTION

Complex systems design involve a varied set of modeling experts from different business areas. These designers can be located in distant geographical areas, as it is the case in distributed collaborative development in big software companies. Several approaches have been developed to face complex systems' modeling. The most used one is the multi-modeling approach (Boronat et al., 2008). It consists in elaborating separate partial models that correspond to different business views on the system (Hilliard, 2001)(Boulanger et al., 2009). This approach helps designers focus in isolation on different parts (partial models) of the system. However, at some point, it is mandatory to construct a global model to understand and effectively exploit the whole system. Our approach sets a matching process as presented in (El Hamlaoui et al.,). It allows the creation of a global view of the system through a composition based on aligning partial models. Our matching process first identifies correspondences between elements of metamodels (meta-elements). We call those correspondences HLC - High Level Correspondences- then, the process generates semi-automatically correspondences between models' elements (LLC - Low Level Correspondences). HLCs and LLCs are stored respectively in M2C (Model of correspondences between metamodels) and M1C (Model of correspondences between models). Thus, the global model is the network of partial models elements, linked thanks to the established model of correspondences.

Partial models may evolve during the system life cycle. As their design was made separately by different designers, their evolution within a system may occur in an uncoordinated manner. Changing one or several elements, involved in a correspondence, may cause the inconsistency of the global model. Our current objective is to ensure the consistency of M1C by re-evaluating LLCs after the evolution of each partial model. One possible approach would be to relaunch the matching process after each evolution. This solution is not optimal as it reproduces the model of correspondence "from scratch" ignoring the previously created matches. Furthermore, no trace of the changes will be kept.

Our proposition takes part in the GEMOC initiative (GEMOC, 2017). In this paper, we present an overview of the matching approach (detailed in (El Hamlaoui et al.,)) and present thereafter the consistency management which is the added value and the core of this current work. Its role is to automatically detect changes from many heterogeneous models and treat their impacts automatically (in some cases semi-automatically) in order to ensure the coherence of the system. The process of maintaining consistency is automatically activated when the matching process produces M1C. It takes as input all the partial models of the application domain and the model of correspondence M1C as presented in Figure 1. Our approach does not concern intramodel changes (elements changes that impacts other elements of the same partial model); it is up to the designers of partial

El Hamlaoui, M., Bennani, S., Nassar, M., Ebersold, S. and Coulette, B.
A MDE Approach for Heterogeneous Models Consistency.
DOI: 10.5220/0006774101800191
In *Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2018)*, pages 180-191
ISBN: 978-989-758-300-1

models to manage the internal repercussions of changes made on their models and to ensure their validity. Thus, apart from the added elements, only changes (modification and deletion) that have an effect on the elements of the M1C are treated.

The rest of this paper is organized as follows: Section 2 introduces the Conference Management System that was chosen to apply our approach. Section 3 presents a general view of the matching approach whereas section 4 describes in detail the consistency management approach. Section 5 presents the related work. We conclude this paper by some perspectives and a conclusion in section 6.

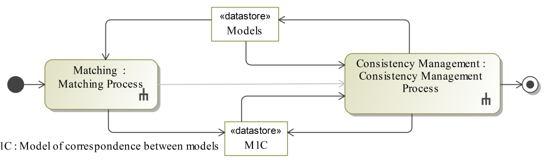


Figure 1: Overall process of our approach.

2 RUNNING EXAMPLE: CMS (CONFERENCE MANAGEMENT SYSTEM)

Production and reviewing of documents are widely used in various contexts such as project management, software development, conference management, etc. A Conference Management System (CMS) is a system designed to automate the main functions needed for the management of a scientific conference, namely: call for papers, papers submission, papers assignment for evaluation, notification of the final decision, registration, etc. Although, this system is not very complex, we had chosen it for two reasons; firstly, it is well known by almost every researcher; secondly and most importantly, it involves different designers, working with different points of view.

We represent this system by three models: a software design model, a business process model and a persistence model. Due to space limit, we refer to (Author, 2017) where we provide and present in detail the three models and their respective metamodels.

3 MATCHING APPROACH

Our matching approach consists in analyzing input models (and their respective metamodels) in order to identify correspondences that exist among them. Correspondences are stored into a model of correspondences (M1C) conforming to a metamodel of correspondences (MMC). As mentioned earlier, we only consider intermodels correspondences. So, correspondences between elements of the same partial model are out of the scope of our research study. We present below the elaboration of M1C as well as the proposed iterative matching process.

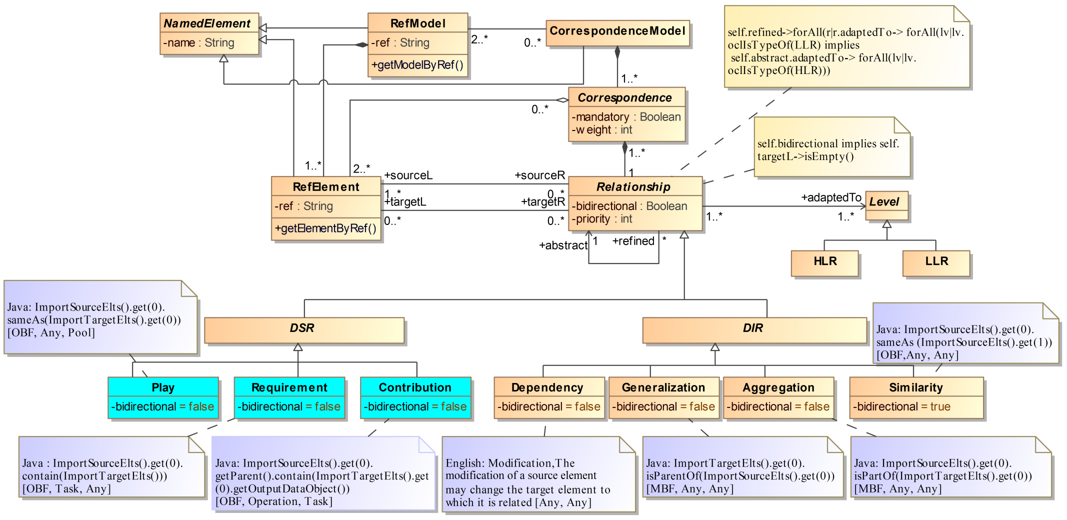
3.1 Metamodel of Correspondences

MMC represented in Figure 2 identifies the different concepts through which M1C is created. *CorrespondenceModel* contains correspondences established between at least two (meta)-elements (*RefElement*) from different (meta)models through their *references*. We store references as String value to encapsulate information about both their source model and source metamodel. The *Correspondence* meta-class has two attributes that works for the consistency management process : the *mandatory* attribute specifies whether a correspondence is obligatory to the studied system or not. The *weight* attribute represents the weighting coefficient associated to each correspondence. These two attributes will be explained in section 4.

The *correspondence* meta-class is composed of at least two referenced elements and the *Relationship* that connects them. The *bidirectional* attribute specifies whether the relationship is bidirectional or not. If the relationship is bidirectional, the concerned (meta-)elements are all source ones and there is no (meta-)element of type target, as specified by the OCL rule :*self.bidirectional implies self.target* $\rightarrow$ *isEmpty ()*. The priority attribute, which is also used in the consistency management process, granted a priority value to each type of relationship.

Relationship is an abstract generalization of *DIR* (Domain Independent Relationship) and *DSR* (Domain Specific Relationship) meta-classes. The specialization of the first one allows representing generic relationships that are generic and independent from the domain of application. However, these relationships may be insufficient for a given domain. In this case, it is possible to add specific relationships by a specialization of the DSR meta-class. For the CMS, we had to add the following three DSRs to describe the whole system: Play, Requirement and Contribution. The first one highlights the role played by the participants of a conference in the business process. The second allows us to know the required input for a task. The third identifies the operations that contribute to the success of a task.

The abstract meta-class *Level* is associated to a relationship to describe whether it represents a *High Level Relationship* (HLR) or a *Low Level Relationship* (LLR) or both. *HLR* allows representing relationships



self.refined->forAll(r|r.adaptedTo-> forAll(lv|lv.oclIsTypeOf(LLR) implies self.abstract.adaptedTo-> forAll(lv|lv.oclIsTypeOf(HLR)))

self.bidirectional implies self.targetL->isEmpty()

Java: ImportSourceElts().get(0).sameAs(ImportTargetElts().get(0)) [OBF, Any, Pool]

Java: ImportSourceElts().get(0).sameAs (ImportSourceElts().get(1)) [OBF,Any, Any]

Java : ImportSourceElts().get(0).contain(ImportTargetElts())) [OBF, Task, Any]

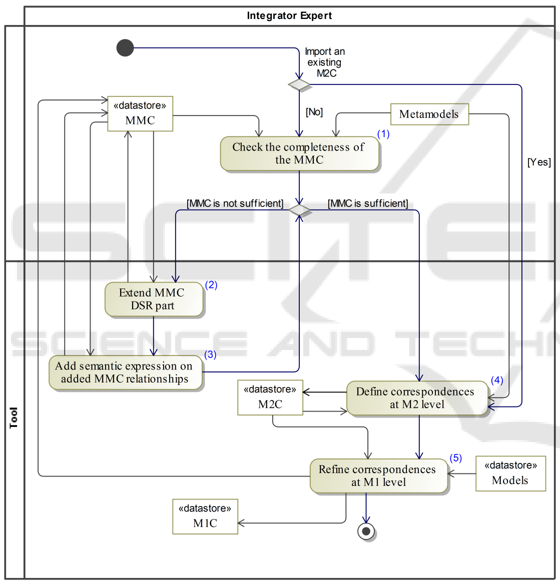
Java: ImportSourceElts().get(0).getParent().contain(ImportTargetElts().get(0).getOutputDataObject()) [OBF, Operation, Task]

English: Modification,The modification of a source element may change the target element to which it is related [Any, Any]

Java: ImportTargetElts().get(0).isParentOf(ImportSourceElts().get(0)) [MBF, Any, Any]

Java: ImportSourceElts().get(0).isPartOf(ImportTargetElts().get(0)) [MBF, Any, Any]

Figure 2: MMC with semantic expressions.



MMC : Metamodel of correspondence
DSR: Domain Specific Relationship
M2C : Model of correspondences between metamodels
M1C : Model of correspondences between models

Figure 3: Matching sub-process.

that are used in correspondences at metamodel level, whereas *LLR* represents relationships that are used in correspondences at model level. The OCL rule: *self.refined* $\rightarrow$ *forAll (r — r.adaptedTo* $\rightarrow$ *forAll (lv — lv.oclIsTypeOf (LLR) implies self.abstract.adaptedTo* $\rightarrow$ *forAll (lv — lv.oclIsTypeOf (HLR)))* specifies that any type of relationship usable at the HLR level is also usable at the LLR level. The opposite is not true. Other annotations are associated to relationships. They are created during the mathing process and they contain their semantics expressions which will be used during the selection operation (section 3.4). For example for the Similarity relationship, the semantic explains that this relationship is used in a correspondence involving two elements (recovered by the importSourceElts() operation) with indifferent types (Any, Any). It is described by an expression in Java. The condition explicitly states, using the sameAs function, that the related elements are similar.

3.2 Matching Process

The model of correspondences cannot be constructed in a monolithic manner. It follows a process that we call matching process. Figure 3 shows one iteration of the proposed process. The process involves two stakeholders, namely, an integrator expert who is the supervisor of the application domain, and a tool that assists the supervisor and enacts the automatic parts of the process.

Firstly, the process takes as input the various metamodels and the kernel of the metamodel of correspondences. Subsequently, the supervisor verifies if the MMC contains all needed relationships to set up correspondences between partial (meta)models. If he/she assumes that the proposed relationships are not sufficient to describe the given domain of application, the DSR meta-class of MMC is specialized. In the case of CMS, we added three types of relationships: Play, Requirement and Contribution. The third activity of this process enriches the MMC with a Semantic Expression (SE) for each relationship. For this purpose, we proposed a Semantic Expression DSL (El Hamlaoui et al., 2015) that is woven with the MMC as annotations. The advantage of using this DSL is primarily to have a structured common definition of each relationship. Secondly, it helps build M2C in an assisted way using information on the connected elements types. Thirdly, it helps filter out the correspondences in the selection step in order to keep only correspondences that match the semantics of their relationship.

Once the MMC is specialized, the matching operation begins, the supervisor identifies correspondences between meta-elements in order to produce the model of correspondences called M2C. M2C (Model of correspondences between metamodels) stores High Level Correspondences that contain meta-elements linked by High Level Relationships. Figure 4 summarizes the thirteen HLCs defined for CMS. We cite two examples: A *Contribution* correspondence links the meta-element *Task* from the *Business Process* metamodel to the meta-element *Operation* from the *Software Design* metamodel, since an operation can potentially contribute to the achievement of a task. A *Similarity* correspondence is established between the meta-elements *Table* and *Entity* in one side and between the meta-elements *Property* and *Column* on the other side.

HLCs are then refined in order to produce LLCs. Our developed tool produces them semi-automatically by performing a reproduction operation on the M2C followed by an operation of selection.

3.3 Reproduction

This operation is a homomorphism (structural preservation from one algebraic structure to another one) between correspondences in M2C and M1C. It duplicates all correspondences defined at the metamodel level into the model level. So, there will be as many potential LLCs for a given HLC as Cartesian product of instances of meta-elements involved in the HLC. This operation limits the generation of correspondences to elements whose type participates in a HLC.

Even if the contextual information helps avoiding the creation of correspondences between elements of types that do not match (e.g., an Operation and a Field) it does not guarantee that all generated correspondences are semantically correct.

3.4 Selection

This operation consists in filtering out correspondences produced by the reproduction operation in order to keep only those who are valid, with respect to the semantic expression associated to their relationship, and filter out the incorrect ones. For relationships with informal expression (in natural language), the supervisor decides whether or not to keep the correspondences depending on the expression associated to their relationships. Considering the relationships with a formal expression, their expressions (represented as a note in Figure 2) have to be executed. Execution of body's expressions requires an interpreter of the language in which the expression is written (a Java Virtual Machine in our case). Figure 5, illustrates the M1C model of CMS obtained when the expert manually does the selection phase. In parallel, the expert has performed the matching process through the tool and compared the M1C that he has produced manually to the one generated by the tool. This comparison concerns only three relationships (i.e. similarity, aggregation and contribution) as they are the only ones implemented on the tool. For example, executing the following sameAs method: "Author".sameAs("AuthorTable") returns true. Thus, the tool keeps the correspondence involving the two elements. Similarly, the execution of these two code snippets: "firstName".isPartOf("fullName") and "lastName".isPartOf("fullName") return true. This leads to keeping the correspondence with the aggregation relationship between the two source elements (firstName, lastName) and the target element (fullName). One the tool generates the M1C, the expert evaluates the accuracy of produced LLCs. Figure 6 shows the results of comparison between the M1C obtained manually and the one generated through the tool. On the line axis, we present for each type of relationships, LLCs considered valid according to the tool (true) and those considered false. Then each bar is split to distinguish LLCs truly validated by the expert (true expert) from invalid ones (false expert). Considering precision and recall about these results, we have obtained a precision of 86% for the similarity relationship, 50% for the aggregation and 70% for the contribution. While the value of recall is respectively 75%, 75% and 100% for the similarity, aggregation and contribution relationships. The following section will discuss the consistency management based on the manually produced model of correspondences presented in Figure 5. The seventeen LLCs produced were numerated to facilitate their exploitation on the next section.

4 CONSISTENCY MANAGEMENT APPROACH

Since models evolution is generally not coordinated between partial models designers, each model may evolve independently. So, it is very tedious to rerun the matching process after each change due to the human effort required in the matching activity and the lack of changes tracking.

Our approach provides a consistency management

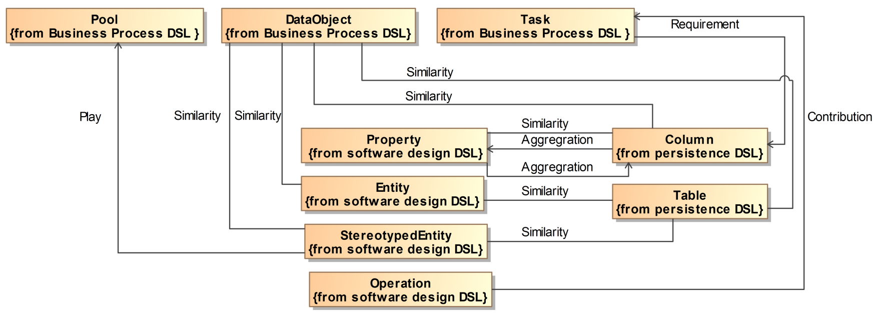

Pool {from Business Process DSL }
DataObject {from Business Process DSL}
Task {from Business Process DSL }
Requirement
Similarity
Similarity
Play
Similarity
Similarity
Contribution
Property {from software design DSL}
Similarity
Aggregation
Aggregation
Column {from persistence DSL}
Entity {from software design DSL}
Similarity
Table {from persistence DSL}
StereotypedEntity {from software design DSL}
Similarity
Operation {from software design DSL}

Figure 4: Example of M2C Model for the CMS system.

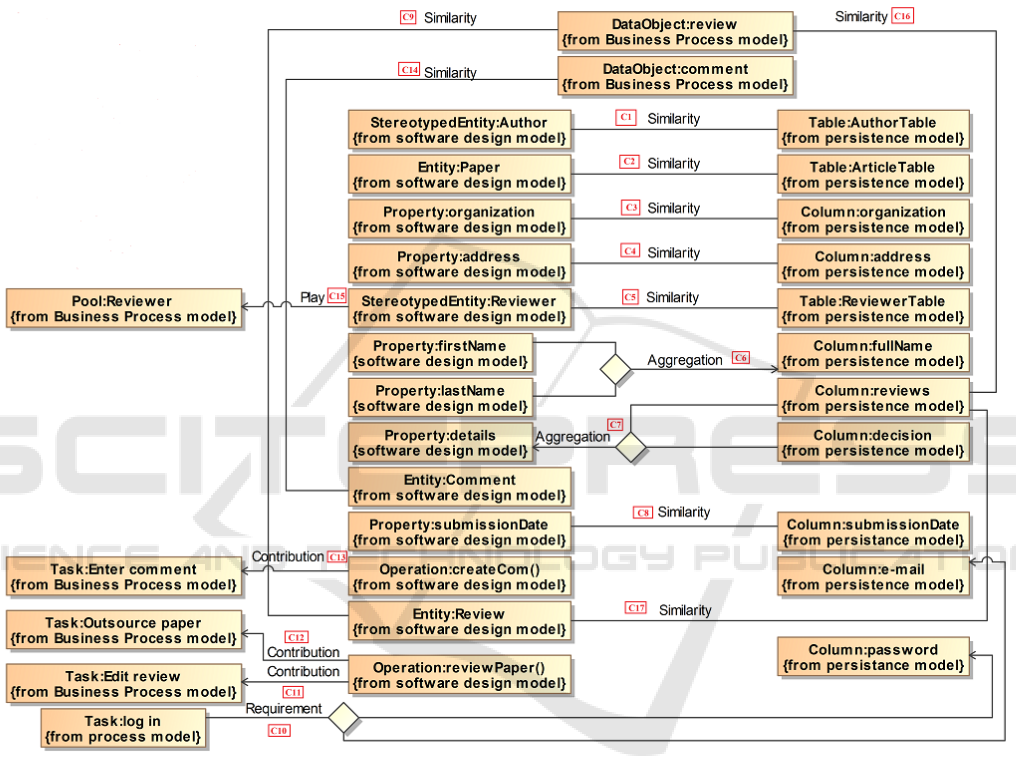

C9 Similarity
DataObject:review {from Business Process model}
Similarity C16
C14 Similarity
DataObject:comment {from Business Process model}
StereotypedEntity:Author {from software design model}
C1 Similarity
Table:AuthorTable {from persistence model}
Entity:Paper {from software design model}
C2 Similarity
Table:ArticleTable {from persistence model}
Property:organization {from software design model}
C3 Similarity
Column:organization {from persistence model}
Property:address {from software design model}
C4 Similarity
Column:address {from persistence model}
Pool:Reviewer {from Business Process model}
Play C15
StereotypedEntity:Reviewer {from software design model}
C5 Similarity
Table:ReviewerTable {from persistence model}
Property:firstName {software design model}
Aggregation C6
Column:fullName {from persistence model}
Property:lastName {software design model}
Column:reviews {from persistence model}
Property:details {software design model}
Aggregation C7
Column:decision {from persistence model}
Entity:Comment {from software design model}
Property:submissionDate {from software design model}
C8 Similarity
Column:submissionDate {from persistance model}
Task:Enter comment {from Business Process model}
Contribution C13
Operation:createCom() {from software design model}
Column:e-mail {from persistence model}
Entity:Review {from software design model}
C17 Similarity
Task:Outsource paper {from Business Process model}
C12 Contribution
Column:password {from persistance model}
Task:Edit review {from Business Process model}
Contribution
Operation:reviewPaper() {from software design model}
C11
Requirement
Task:log in {from process model}
C10

Figure 5: M1C Model for the CMS system.

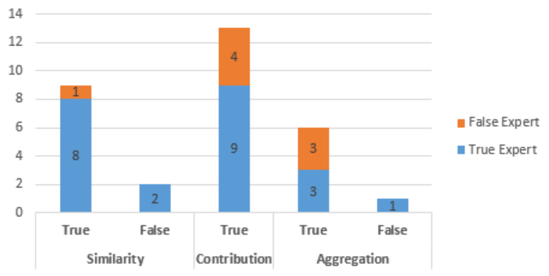


Figure 6: Comparison between the M1C produced manually and the one generated via the tool.

process. This process is automatically activated using the Observer pattern (Vlissides et al., 1995) at the end of the matching process. It takes as input the system's partial models and the model of correspondences between them (M1C) and it follows six steps as shown in Figure 7 (change detection, change analysis, cycle management, change scheduling strategy, change prioritization and change processing). The first step requires continuous monitoring while the others are triggered successively on the expert's demand. This process is carried out by a developed tool and imply the supervisor's intervention in phases that require a human expertise or configuration. Throughout this section, we are going to detail these six steps.

4.1 Changes Detection

Changes are detected when they occur through the Observer pattern model instead of differencing

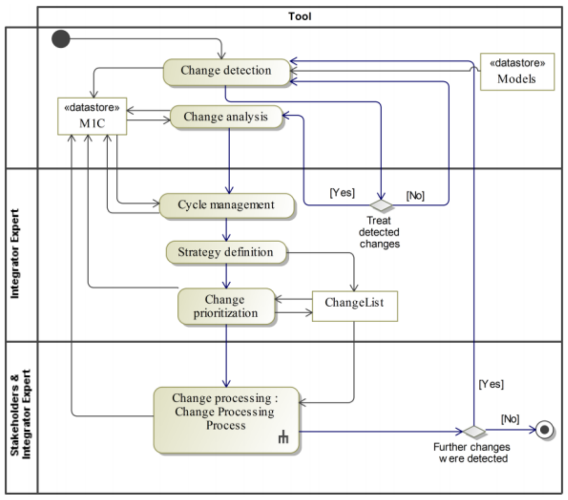


Figure 7: Consistency management sub-process.

techniques as done in EMFCompare (Brun and Pierantonio, 2008) that provides a generic algorithm for calculating differences between two versions of a model. The problem with these techniques is threefold. Firstly, the whole model must be parsed in order to check if an element has changed between two versions of a model, based on distance computing techniques. Secondly, there is a restriction on the number of models because comparison can be done with only two models of a given business domain. Thirdly, there is a waste of memory since they require keeping in memory the previous version of the input model as well as the current one.

Changes detected by the implementation Observer pattern are subsequently added to M1C using the MMC meta-classes *History, DiffElt, AddedElt, DeletedElt, ModifiedElt* (part 1 of Figure 8). *History* is used to keep track of applied changes. *DiffElt* allows to record the trace of evolved elements. It has two attributes. The first attribute contains the change classification type. The second attribute contains the reference of the element constructed from the element's name, its meta-element and the model's name. *DeletedElt* represents an element that no longer exists in the original model but that is maintained for tracing purpose. *AddedElt* and *ModifiedElt* respectively represent newly added element and model element that has undergone a modification. The extended MMC defines moreover a concept that represents the core of the observer pattern (part 2 of Figure 8). The *Observer* meta-class specifies the model's element to be observed. It is a generalization of the subject meta-class which has three methods. Two of them (*attach* and *detach*) allow to fix or detach an observer object from a model element. The third method (*notify*) makes it possible to notify the M1C of the changes that have taken place. The *update* method of the meta-class Observer is used during the phase of changes processing in order to maintain the consistency of domain's models. The third part of Figure 8 (i.e. the *impact* meta-class) defines the *impact kind* of each change and the *solution* for each change. A change on an element may affect directly or indirectly elements that are linked to this element by a correspondence as we will see in section 4.2.

The column change detection of Table 1 summarizes the evolutions perfomed on the CMS models. These changes are automatically detected via the tool (with a continuous monitoring) and added to the M1C. For the CMS, we consider that the process architect has added the two roles "chairman" and "author" to the business process model and that the software architect and the process architect have removed the following elements: *Property:phoneNumber*, *Entity:Comment* and *Task:outsource paper*. We also consider that the software architect has renamed the operation *createCom()* and replaced the *Entity:Review* element with *Entity:Note* element.

4.2 Changes Analysis

This analysis includes defining the type of change and the M1C elements that may be affected by this change. This is possible thanks to the extension of the MMC, which makes it possible to find for a modified element the correspondence(s) to which it belongs and thus to find the element(s) that may be affected. Directly impacted elements are elements directly related to a modified element, whereas indirectly impacted elements are elements that may be affected via a cascading effect.

Suppose we have two correspondences (Figure 9), the first linking an element A to an element B by the relationship *Rx* and the second one relating the element B to an element C by the relationship *Ry*. If element A is modified, element B can be influenced directly by this change, whereas element C may be affected indirectly via the cascading effect.

In this phase, we also classify changes in two categories: the automatic mode for added and deleted elements and the monitored mode for the modified elements. In this latter when an element has changed, the correspondence must be assessed in terms of the semantics of its type of relationship. According to (Cicchetti and Ciccozzi, 2013), when the relationship type semantics comes into play, version management problems become more complex and can not be processed automatically. In other words, the automation support is reduced and the changes likely involve human assistance to decide the impact of the change.

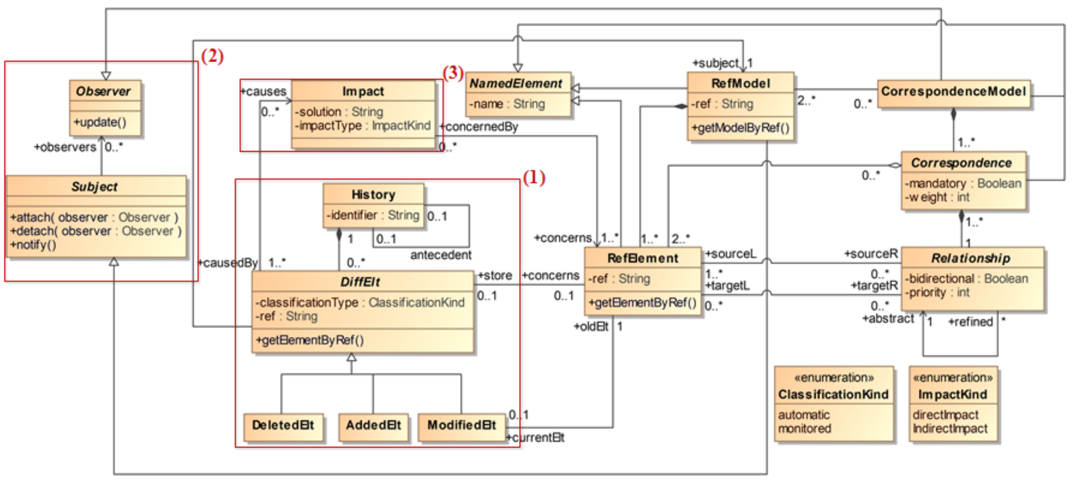


Figure 8: Extension of the Correspondence Metamodel MMC to handle consistency management.

Table 1: Steps of the Consistency Management Process on the CMS system.

Model's Element	**Change Detection**	**Change Analysis**		**Change Priorization**		**Change Processing**
	Type	Classification	Influenced Elts.	Weight	Order	
Pool:Chairman	Add	Automatic	-	0	4	**Matching process**
Pool:Author	Add	Automatic	-	0	3	**Matching process**
Task:Outsource paper	Del	Automatic	*Operation: reviewPaper()(DirectImpact) Task:Edit review(IndirectImpact)*	8	1	**Restoring the deleted element**
Property: phoneNumber	Del	Automatic	-	-	-	-
Entity:Comment	Del	Automatic	*DataObject: Comment(DirectImpact)*	1	2	**Deleting the correspondence**
Old *Operation: createCom()* New *Operation: createComments()*	Mod	Monitored	*Task: Enter comment(DirectImpact)*	4	6	**Maintaining the correspondence**
Old *Entity:Review*	Mod	Monitored	*DataObject: Review(DirectImpact), DataObject: Review(IndirectImpact)*	6	5	**Modifying the correspondence's end element** (C9)
New *Entity:Note*			*Column: reviews(DirectImpact), Property: details(IndirectImpact)*			**deleting the correspondence** (C16)

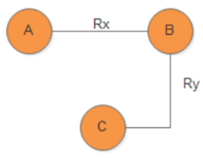


Figure 9: Example of cascading effect.

Column *change analysis* of Table 1 shows the information that are automatically added to the M1C during this phase. The added elements are classified in the automatic mode since adding them does not affect the consistency of the M1C. Deleted items are also classified in the automatic mode.

Deleting the *Property:phoneNumber* element has no effect on M1C since it does not participate in any correspondence. Deleting the *Entity:Comment* element has a direct impact on the *DataObject:comment* (linked via correspondence C14 as shown in Figure 5) while deleting the *Task:Outsource paper* element has a direct influence on the *Operation:reviewPaper()* element (correspondence C12) and an indirect influence on the *Task:Edit review* element (correspondence C11).

Modified elements are classified in guided mode. The modification of *Operation:CreateCom()* element has a direct influence on the *Task:Enter comment* element (via correspondence C13), whereas the change on the *Entity:Review* has a direct influence on the *DataObject:review* (correspondence C9) and *Column:reviews* (correspondence C16) and an indirect influence on the *DataObject:review* (correspondence C17) and *Property:details* (correspondence C7).

4.3 Cycle Management

Once changes and their direct or indirect impacts are detected, the tool catches automatically the cycles of cascading effects. Then the expert chooses to delete one of the correspondences in order to break the cycle. Let's take the example presented in Figure 10, we have three correspondences. The first one relates an element A to an element B, the second one connects element B to an element C and the third one relates C to A. If the element A is modified, the element B can be directly influenced by this change, which indirectly influences the element C. In the case where the element C is the one that causes the modification of the Element A, we will have a cascading cyclical effect. The various cascading cyclical effects are reported to the experts to decide to delete one of these correspondences in order to break the cycle.

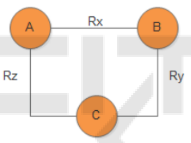


Figure 10: An example of the cascading cyclical effect.

4.4 Change Scheduling Strategy

This step aims at producing an ordered list of changes. We propose two strategies for changes ordering: classification-based strategy and impact-based strategy.

The classification-based strategy consists of creating a list of changes that contains, in order, changes that are classified in automatic mode followed by those in monitored mode. For example, by choosing this strategy, the list of changes produced in the CMS is as follows: *Pool:Chairman*, *Pool:Author*, *Task:Outsource paper*, *Property:phoneNumber*, *Entity:Comment*; *Operation:createComments()* and *Entity:Note*. The second strategy that we propose creates an ordered list depending on the type of impact of each change. For example, the expert may start by processing the changes that have both direct and indirect impacts on other elements and leave changes that have only direct impacts on other elements to the end. For example, by choosing this strategy, the list of produced changes is as follows: *Entity:Note*, *Task:Outsource paper*, *Pool:Chairman*, *Pool:Author*, *Property:phoneNumber*, *Entity:Comment*, *Operation:createComments()*.

These two scheduling strategies work for changes that have different modes of change or different types of impact. In the next section, we will see how to classify changes that have the same type of impact or the same mode of classification.

4.5 Change Prioritization

Changes processing order has an impact on the system and its consistency. That's why we attribute a weighting coefficient to each correspondence so we can determinate a prioritized processing order. Taking the example of Figure 10, assuming that elements A and C have been modified, the question to be raised is which change to treat first, knowing that both have an influence on the same element B? and will the two changes be treated?. Prioritization allows to answer these two questions. We first deal with the change that has the highest weighting coefficient, next change must then take into account the impact of the previous changes. In other words, if the change of the element A requires the modification of the element B, the element C can modify the element B only if the correspondence linking the element A and B has not been changed semantically. The order of processing is defined by calculating the weighting coefficient of the correspondence. This coefficient is calculated by the following formula:

$$\begin{aligned} weight &= \sum_{k=0}^{n}(DirectlyAffectedElement_k * priority) \\ &+ \sum_{k=0}^{n}(IndirectlyAffectedElement_k * priority) \end{aligned}$$

Where the term priority is the level of priority given to each type of relationship. We consider that the supervisor gives level 1 to the relationship *Similarity*, level 2 to the *Aggregation*, level 3 to the *Play* relationship, level 4 to the *Contribution* and level 5 to the relationship of *requirement.*

Sub column *order* of column *change prioritization* of Table 1 illustrates the result of the change prioritization phase according to the strategy chosen in the previous step (in our case, automatic mode first) and to the calculated coefficients.

When several changes have the same weighting coefficient, it is up to the expert to decide on the processing order, except for the addition type changes for which there is no order preference given that all added elements are processed at the end of the change processing by using the matching process. Let's detail the weighing coefficient calculation for the highest change (deletion of *Task: OutsourcePaper*) : This element is directly linked to *Operation: ReviewPaper()* via a *Contribution* link (priority 4) and indirectly linked to *Task:Edit Review* via a *Contribution* link also. The weighting coefficient is thus 4*1+4*1=8.

4.6 Change Processing

In this step, M1C and the partial models may be modified to take account of the changes that have been detected. Figure 11 presents the change processing process. Changes categorized in automatic mode are processed automatically.

In case where an element has been added, the matching process is restarted at the end of the change process to handle the added elements all at once. In case an element has been deleted, all correspondences involving this element become orphaned. An orphaned correspondence is a correspondence for which one of its ends - a model element - is missing. When a correspondence becomes orphaned, the expert checks if it is mandatory for the system concerned (true in the mandatory attribute). In positive case, the deleted item is restored, otherwise the correspondence is deleted from the model of correspondences.

Concerning the second type of changes, namely changes occurring in a monitored mode, they are managed semi-automatically. The correspondence is maintained if, after the change of one of its ends, it remains correct regarding the semantic associated to its type of relationship. Otherwise, when an element is modified, it is necessary to modify each of the elements tied to it since this modification is possible. When it is impossible to modify an element, the correspondence is deleted if it is not mandatory (mandatory = false). Otherwise, a group decision making, involving the expert with partial model designers, takes place to decide whether to modify the concerned element or the element at the other end of the correspondence.

For the CMS example, we proceed following the column *order* of Table 1, the first removed element is restored because it is used in the correspondence C12 (see Figure 5) and it is a compulsory correspondence (mandatory = true). Correspondence C14 involving the second element is deleted, since it is an orphan correspondence and it is not mandatory to the system consistency. The matching is invoked at the end of the process to search for possible correspondences for the third and the fourth element. Two correspondences using the *play* relationship are thus created (C17 and C18). The fifth element is involved in two correspondences C9 and C16. The first one links it to the *DataObject:review* element and the second to the *Column:reviews* element. The element of the opposite end of the first correspondence is modified since the correspondence is no longer correct. Concerning the second correspondence, since the opposite element cannot be modified (because of the choice to prohibit the modification or the deletion of any element of the persistence model) and the correspondence is not obligatory, it is then deleted.

Table 2 summarizes the change processing on the CMS. Lines in bold indicate changes that have been made to M1C of Figure 5. The underline lines (correspondences: C17 and C18) are the added elements, those in dotted lines (correspondences: C12 and C13) are the changed elements. Deleted elements are barred (C14 and C16).

5 RELATED WORK

A number of approaches described in the literature deal with view-based complex systems' design through models matching. As this paper mainly addresses the consistency management due to an evolution, in this section, we put the focus on the subset of these approaches that support one or several aspects of model evolution issue. To do that, we have studied a set of representative approaches, namely Edapt(Williams et al., 2012), previously called COPE(Herrmannsdoerfer et al., 2008), EMFMigrate (Di Ruscio et al., 2011), Model federation (Guychard et al., 2013) and the one developed by Cicchetti and al. (Cicchetti et al., 2008). To lead this study we have defined and used the following criteria that we consider relevant in a multi-modeling environment: heterogeneity, number of input artifacts and their types, mechanism of change detection, adopted support of changes' classification, and migration rules. These criteria are defined as follows:

- Heterogeneity: expresses if the considered approach takes into account heterogeneous artifacts. We consider that two artifacts are heterogeneous if their modeling languages are themselves different,
- Number of input Artifacts: indicates the maximum number of input artifacts allowed by the approach,
- 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,
- Change Detection: assesses how an approach proceeds to detect the elements of artifacts that have undergone an alteration,

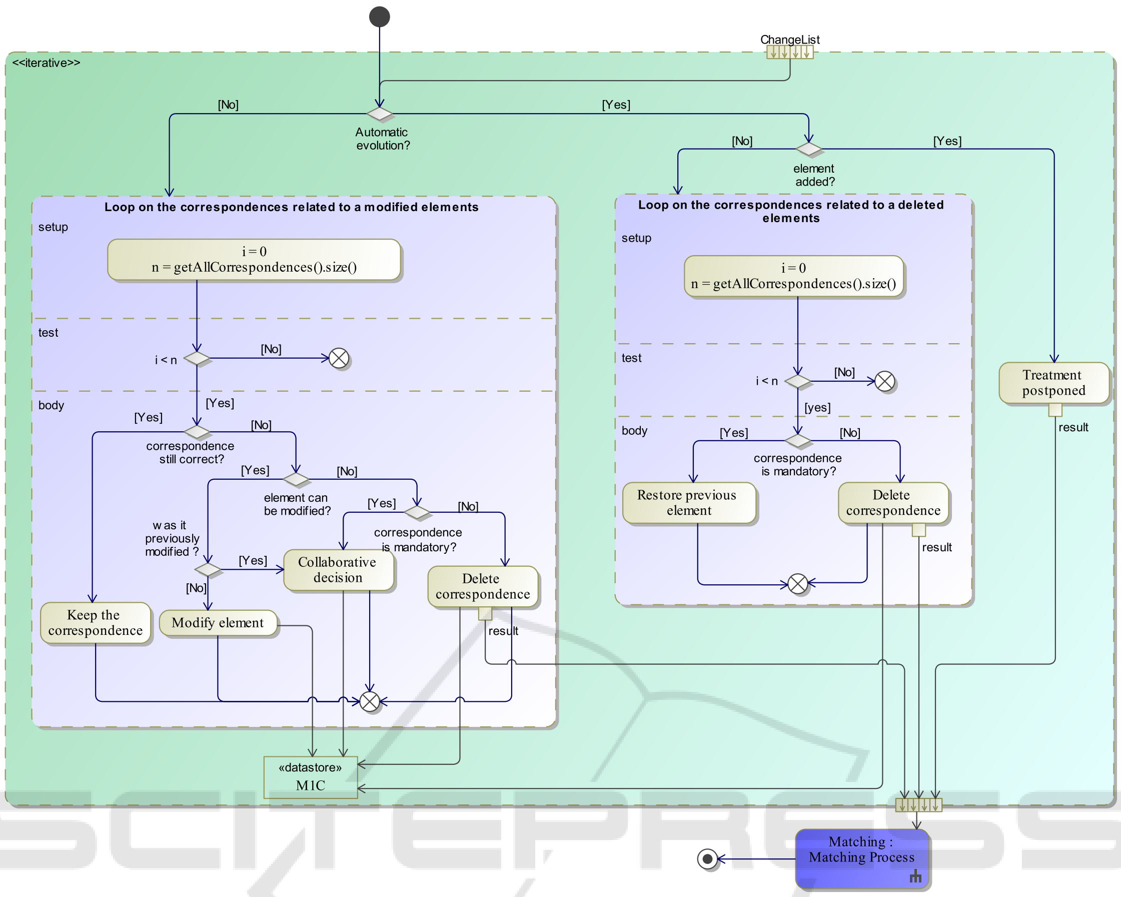


Figure 11: Change processing steps.

Table 2: CMS system's M1C model after models evolution and Consistency Management Process conduct.

Correspondence		**Relationship's Type**		**Partial Models' Elements**		
Name	Man-datory	Name	Prio-rity	Software Design Model	Business Process Model	Persistence Model
C1	False	*Similarity*	1	*StereotypedEntity:Author*		*Table:AuthorTable*
C2	False	*Similarity*	1	*Entity:Paper*		*Table:ArticleTable*
C3	True	*Similarity*	1	*Property:organization*		*Column:organization*
C4	True	*Similarity*	1	*Property:address*		*Column:address*
C5	False	*Similarity*	1	*Entity:Reviewer*		*Table:ReviewerTable*
C6	False	*Aggregation*	2	*Property:firstName* *Property:lastName*		*Column:FullName*
C7	False	*Aggregation*	2	*Property:details*		*Column:reviews* *Column:decision*
C8	True	*Similarity*	1	*Property:submissionDate*		*Column:admissionDate*
C9	**False**	***Similarity***	**1**	***Entity:Note***	***DataObject:notice***	
C10	True	*Requirement*	5		*Task:logIn*	*Columns:password, e-mail*
C11	True	*Contribution*	4	*Operation:reviewPaper()*	*Task:Edit review*	
C12	**True**	***Contribution***	**4**	***Operation:reviewPaper()***	***Task:Outsource paper***	
C13	**True**	***Contribution***	**4**	***Operation:createComments()***	***Task:Enter comment***	
~~**C14**~~	~~**False**~~	~~***Similarity***~~	~~**1**~~	~~***Entity:Comment***~~	~~***DataObject:comment***~~	
C15	False	*Play*	3	*StereotypedEntity:Reviewer*	*Pool:Reviewer*	
~~**C16**~~	~~**False**~~	~~***Similarity***~~	~~**1**~~		~~***DataObject:review***~~	~~***Column:reviews***~~
C17	False	*Play*	3	*StereotypedEntity:Author*	*Pool:Author*	
C18	False	*Play*	3	*StereotypedEntity:Chairman*	*Pool:Chairman*	

- Classification Support: indicates whether the approach supports a classification of changes in order to assign to each kind of change a particular action. It is interesting to take this criterion into account, because the classification of changes allows the automation of the whole evolution management process or at least a part of it,
- Migration Rules: describes the language in which the migration rules are implemented. It is with this language that the treatment of evolution is realized.

Table 3 presents a synopsis of the studied approaches, based on the established criteria. By analyzing it, we can deduce that the consistency management process has not reached a sufficient maturity yet. Firstly, the studied approaches do not define any classification support, a factor that we consider mandatory to automatically manage changes and their impacts on models, through predefined actions. Secondly, for a complex change which affects several elements at the same time, multiple rules are likely to be produced. Choosing the appropriate adjustment to run requires rules' customization and also some technical background knowledge. Thirdly, most of the approaches discussed above, except Model Federation and Cicchetti and al., focus on model evolution as a result of adaptation of their corresponding metamodels (co-evolution) to preserve the conformity relationship. That is to say that these approaches emphasize the vertical level (co-evolution) without considering horizontal evolution (between models). It is within the scope of this latter that models synchronization is based. Fourthly, EMFMigrate and Cicchetti and al. require a specific difference metamodel and it is up to the stakeholder to detect the changes and represent them in a difference model. Fifthly, the model federation approach responds to our problem by exploiting correspondences established for the consistency management. However, it does not propose a mechanism for managing traceability of changes. Moreover, the approach is based essentially on the modification change of type. Therefore, it does not manage impacts due to the addition or deletion of model elements. For the modification, the synchronization cannot be applied without defining a master and a slave model.

To sum up, the approaches presented above do not consider or respect all of the criteria described above and thus do not fully address some important aspects of model evolution. This is because they do not exploit previously established correspondences to provide a mechanism that ensures the consistency of the overall model.

6 CONCLUSION AND PERSPECTIVES

Our general research work addresses view-based complex information systems design. During the modeling cycle, the description of models evolves frequently due to the emergence of new requirements and constraints. In a multi-modeling environment, several changes can occur on different models of the system. To manage the consistency between these models, we propose to exploit the correspondences model to treat the changes that are identified automatically on partial models in order to maintain the consistency of the interconnected models. Once the changes are identified, the consistency management process proceeds to their classification and the potential impacts are identified automatically as well as the possible presence of cycles. These latters are managed by the expert. Change prioritization is important because without coordination the evolutions treatment could become unmanageable. For this, according to the chosen strategy, a list of changes is generated according to the calculation of weighting coefficients. Finally changes proceed automatically based on a change processing sub-process.

As a proof of concept of our approach we are developing a support tool called HMCS (Heterogeneous Matching and Consistency management Suite). Its role is to provide assistance to expert in the creation of the model of correspondences and the management of the consistency between heterogeneous partial models when they evolve. HMCS is operational but only supports the matching sub-process. This tool, once completed, will allow us to validate our approach and to conduct experiments to verify thereafter its scalability.

We propose two perspectives to our work. The first one concerns the consistency management process. In this process we considered the directly affected elements at the same level as the indirectly affected ones for calculating the weighting coefficients. To strengthen the algorithm, we intend to implement the random walk theory (Fouss et al., 2007) in order to evaluate the probability that an element in the correspondences model (M1C) is impacted by a change either it is directly related or not. So far, our proposal is based on a relatively centralized process, giving a large responsibility to the expert. Big industrial information systems involve several designers working collaboratively. So, the second perspective consists in defining a collaborative process to support the matching and consistency management activities. Indeed, in real complex systems, designers should closely work together to efficiently produce the corre-

Table 3: Comparison of model evolution approaches.

Approaches/Criteria	H	NA	TA	CD	CS	MR
Edapt (Cope)	No	2	M[1]	SA[3]	No	Java (Groovy)
EMFMigrate	No	2	M/T [2]	Manual	No	Specific DSL
Cichetti and al.	No	2	M	Manual	No	ATL[5]
Model federation	Yes	2..n	M	Not defined	No	ATL[5]
Our approach	Yes	2..n	M	Automatic	Yes	M1C+Process

[1] Model [2] Transformation rules [3] Semi-automatic [4] Adaptation [5] ATLAS Transformation Language

spondence model and they also should be collaboratively involved in the management of models' change impacts. We have initiated this work by presenting in (El Hamlaoui et al.,) the design of a complex system via a collaborative process by modeling the Emergency Department (ED) of a hospital.

REFERENCES

Author, A. (2017). Conference management system example.

Boronat, A., Knapp, A., Meseguer, J., and Wirsing, M. (2008). What is a multi-modeling language? In *International Workshop on Algebraic Development Techniques*, pages 71–87. Springer.

Boulanger, F., Jacquet, C., Hardebolle, C., and Rouis, E. (2009). Modeling heterogeneous points of view with modhelx. In *International Conference on Model Driven Engineering Languages and Systems*, pages 310–324. Springer.

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

Cicchetti, A. and Ciccozzi, F. (2013). Towards a novel model versioning approach based on the separation between linguistic and ontological aspects. In *ME 2013–Models and Evolution Workshop Proceedings*, page 60. CEUR-WS.

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

Di Ruscio, D., Iovino, L., and Pierantonio, A. (2011). 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.

El Hamlaoui, M., Coulette, B., Ebersold, S., Bennani, S., Nassar, M., Anwar, A., Beugnard, A., Bach, J., Jamoussi, Y., and Tran, H. Alignment of viewpoint heterogeneous design models: Emergency department case study. In *International Workshop On the Globalization of Modeling Languages (GEMOC) co-located with MODELS 2016*. CEUR Workshop Proceedings.

El Hamlaoui, M., Ebersold, S., Coulette, B., Anwar, A., and Nassar, M. (2015). Maintien de la cohérence de modèles de conception hétérogènes. *Technique et Science Informatiques*, 34(6):667–702.

Fouss, F., Pirotte, A., Renders, J., and Saerens, M. (2007). Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on knowledge and data engineering*, 19(3).

GEMOC (2017). Initiative on the globalization of modeling languages.

Guychard, C., Guerin, S., Koudri, A., Beugnard, A., and Dagnat, F. (2013). Conceptual interoperability through models federation. In *Semantic Information Federation Community Workshop*.

Herrmannsdoerfer, M., Benz, S., and Juergens, E. (2008). Cope: A language for the coupled evolution of metamodels and models. In *International Workshop on Model Co-Evolution and Consistency Management*.

Hilliard, R. (2001). Viewpoint modeling. In *Proceedings of 1st ICSE Workshop on Describing Software Architecture with UML*, volume 7.

Larman, C. (2002). *Applying UML and patterns: an introduction to object-oriented analysis and design and the unified process*. Prentice Hall.

Vlissides, J., Helm, R., Johnson, R., and Gamma, E. (1995). Design patterns: Elements of reusable object-oriented software. *Reading: Addison-Wesley*.

Williams, J., Paige, R., and Polack, F. (2012). Searching for model migration strategies. In *Proceedings of the 6th International Workshop on Models and Evolution*, pages 39–44. ACM.