Heterogeneous models matching for consistency management

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

Heterogeneous models matching for consistency management

Mahmoud EL HAMLAOUI[1,2],
Sophie EBERSOLD[1]
and Bernard COULETTE[1]
[1]IRIT Laboratory, MACAO Team
University Toulouse 2 Mirail
5 Alle Antonio Machado, 31058 Toulouse, France
{mahmoud.el-hamlaoui, sophie.ebersold,
bernard.coulette}@irit.fr

Mahmoud NASSAR[2]
[2]ENSIAS, SIME Laboratory, IMS Team
University of Mohamed V Souissi
BP 713, Agdal Rabat, Morocco
nassar@ensias.ma

Adil ANWAR[3]
[3]EMI, Siweb Laboratory,
University of Mohamed V Agdal
BP 765, Agdal Rabat, Morocco
anwar@emi.ac.ma

***Abstract*—This work is situated in the context of the application of Model Driven Engineering to complex systems view-based modelling. In fact, view-based models - called also partial models - are manipulated by different actors (designers), and are thus generally heterogeneous, that is, described with different DSLs (Domain Specific Languages). Instead of building a single global model, which is not realistic, we propose to organize the different partial models as a network of related models, which provides a global view of the system through a correspondence model. As models are modelled separately by different designers, they also evolve separately that induces a problem of consistency. To solve it, we propose a semi-automatic process based on the correspondence model allowing detecting changes, calculating their impacts, and proposing modifications to maintain the consistency among them. The approach is supported by a tool chain and illustrated by the example of a Bug Tracking System.**

Keywords*—*Heterogeneous models, correspondence model, change processing, consistency.

I. INTRODUCTION

Today, development of complex information 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. To analyze and design such complex systems, one of the most frequently used approaches consists in defining multiple views of the complex system which are defined in different heterogeneous DSLs (Domain Specific Language), each view representing 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. These view-based models -also called partial models- must then be combined in order to represent the complex system. This combination -often called composition- is one of the main issues to which designers must face. A number of approaches investigated the possibility to produce a unique global model by using a merging technique [1] [2] [3]. We have also been working for a while on this topic by elaborating a UML profile called VUML for merging UML partial models [4] [5].
However, it is not realistic to build a unique composed model when the information system become complex since partial models are heterogeneous, that is described by different languages semantically distant. In other words, approaches based on a unique global model are not scalable and thus do not meet our goal. In addition, they do not efficiently support source models evolution since the composed model must be rebuilt each time a significant evolution must be taken in account.

For this reason, we propose an approach consisting in organizing the different partial models as a network of models that provides a global view of the system. This network is composed of models connected via relations that we call ”correspondences”. These correspondences are stored into a ”correspondence model”.
Interconnected models allow the various stakeholders (designers) to work together by manipulating linked elements. In particular, building the correspondence model is a way to solve inconsistencies between separate input models. The correspondence model construction is detailed in [6]. We advocate that such a correspondence model is an important way of expressing semantic rules on the system.

The question that arises now is ”how to manage partial models evolution?” During the modeling or the maintenance phase of a complex system indeed, designers working with specific DSLs according to their viewpoints tend to change the models on which they operate, for example to meet new user requirements. 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.

The approach that we propose is independent from any application domain. So it supports the design of information systems which are particular cases of complex systems.

To sum up, the main advantages of the approach described in this paper are:

- Produce a unique model of correspondences, that relates source models by adding semantic information on the overall system,
- Allows to maintain the consistency of the network of models in case of source models evolution.

In this paper, we present the overall approach with a focus on the consistency management support, especially when partial source models evolve.

The remainder of this paper is structured as follows. Section 2 investigates related works. Section 3 presents the case study that was chosen to illustrate our work. Section 4 describes the matching phase of our approach. Section 5 deals with its consistency management. Section 6 presents the proposed tool prototype and, finally, the paper is concluded in Section 7.

II. RELATED WORKS

A number of approaches described in the literature deal with view-based complex systems design through models matching. As this paper mainly address the consistency maintenance issue, 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[7], previously called COPE[8], EMFMigrate [9], and the one developed by Cicchetti and al. [10].

To lead this study, we have defined and used the following criteria: heterogeneity, number of input artifacts and their types, mechanism of change detection, adopted support of classification, and evolution type. These criteria that should ideally be easy to identify 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,
- Number of input artifacts: 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,
- 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,
- 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,
- Evolution type: characterizes the evolution type: adaptation or co-evolution. The adaptation consists in maintaining the conformity relationship between a model and it's meta-model whereas co-evolution concerns changes at the same level between models, also called model migration.

TABLE I. COMPARISON OF MODEL EVOLUTION APPROACHES

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

[1] Model [2] Transformation rules [3] Semi-automatic [4] Adaptation [5] Co-evolution

Table I presents a synthesis of the studied approaches, based on these criteria. By analyzing it we can deduce that the evolution management process has not yet reached maturity level. Firstly, studied approaches take into consideration only homogeneous models (i.e. derived from the same meta-model). Indeed it is essential to support heterogeneous models so as to tackle real systems. 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. Furthermore, most of the approaches discussed above (except Cicchetti and al.), focus on model evolution 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 adaptation evolution. Yet it is on co-evolution that models synchronization is based.

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 system evolution.

III. ILLUSTRATIVE CASE STUDY

To illustrate our approach, we have chosen a case study based on a real software development project that performs tracking of bugs: BTS (Bug Tracking System). This system can be seen as the software part of an information system that aims to offer to different actors, based on their different status (team leader, developers, testers, ...), the ability to report dysfunctions, comment them, track the status of anomalies, notify collaborators of encountered problems, suggest solutions or possibilities of circumvention, ... The choice of this example seems relevant because it involves different actors, working with different points of view, from the analysis of users requirements to the implementation of the proposed solution.

We consider that in the information system (IS) of bug management, there are three business domains covering various aspects of modelling: user requirements, anomalies and business process. Each business domain is described by a dedicated DSL and is manipulated by actors with specific roles (Figure 1):

- Requirement analyst: Responsible for modelling end user needs (business IS: user requirements). The produced model is expressed through a requirements DSL,
- Software architect: Responsible for modelling anomalies (business IS: anomaly modelling). He creates a model expressed through a specific software design DSL,
- Process engineer: Responsible for bugs tracking process modelling (business IS: process modelling). He

creates a model expressed through a business process DSL.

In the following, we do not consider the concrete syntax of DSLs, so, in the context of this paper, a DSL can be seen as a metamodel defining an abstract syntax.

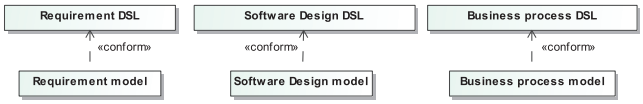


Fig. 1. Global view of the BTS models and DSLs

A. Requirement modelling

To assess the quality and validity of any project, one must ensure that it meets user requirements that are described by the requirements analyst. A requirements DSL inspired from SysML notation [11] (Figure 2) was chosen. A requirements diagram is defined as a canvas containing requirements. Requirements specify, using textual syntax, a capability that a system must satisfy. They are also related to each other or to other model elements using different types of relationships (derived, copy, contains, etc.). The system to build must satisfy requirements described in a model (Figure 3) conform to the previous DSL. For simplicity sake, we have limited the description of BTS to a few requirements. For instance, the requirement identified with id= ”1.1” is related to the declaration of an anomaly; it includes a sub-requirement (id=”1.1.3”) related to the description of the summary of the anomaly, refined in its turn by additional constraints to be respected during the declaration of the anomaly.

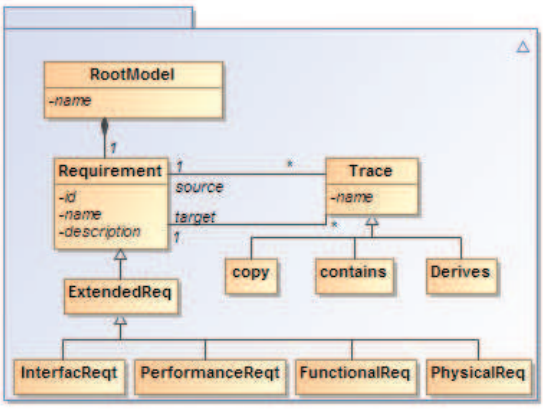


Fig. 2. Extract of the requirement DSL

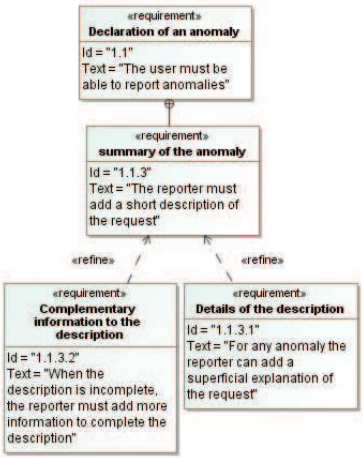


Fig. 3. Snapshot of the BTS requirements model

B. Anomaly management modelling

In Figure 4, we propose a very simple software design DSL to define entities and associations among them. Based on this DSL, we chose an open source software solution in the bug management field called Mantis [12] to represent a software design model. The Figure 5 illustrates a snapshot of such model. The term ”Issue” is used to define an anomaly. An anomaly is characterized by a unique identifier, a set of attributes describing the anomaly namely: category, summary, description, status, steps which led to the anomaly (”stepsToReproduce”) and two types of involved persons with the following roles: ”reporter” and ”assignedTo”. The first role indicates the type of person who reports the anomaly, whereas the second one indicates the type of person to whom the anomaly is affected.

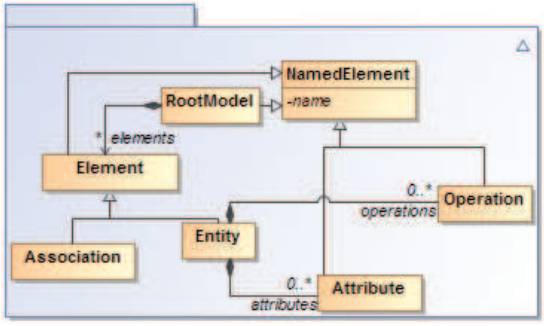


Fig. 4. Extract of the software design DSL

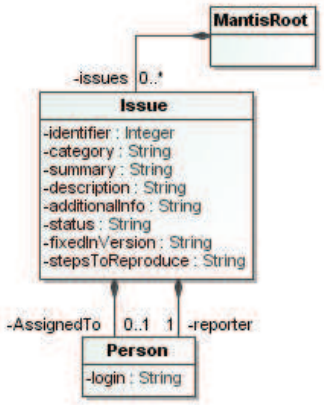


Fig. 5. Snapshot of the software design model

C. Business process modelling

The treatment of an anomaly can be seen as a business process that various collaborators must follow in order to solve the anomaly. This model, complementary from the two others, is necessary to give a behavioural semantics to the bug tracking activity. We suppose that the process engineer uses a business process DSL inspired from BPMN [13]. The DSL (Figure 6) comprises the following concepts: ”lane”, ”pool”, ”flow”, ”task”, etc. A snapshot of the process model expressed in conformity with BPM is presented in Figure 7. Required roles in this process model are ”manager”, ”reporter” and ”developer”. Just after having reported a bug, the ”reporter” must set the status of the anomaly to ”new”. An email is automatically sent to the project manager who has the ”viewer” role as he is not directly involved in the correction of the anomaly. Once the process manager has validated the issue, he must assign it to a ”developer” and change the status to ”open”. Otherwise, if the anomaly is not validated by the process manager, he must reassign it to the ”reporter” to request additional description. Once the ”developer” has corrected the anomaly, he must inform the process manager and change the status of the anomaly to ”Fixed”. The process manager, notified by the change, rechecks the proposed solution and updates the anomaly status to ”closed”, if it has been successfully corrected.

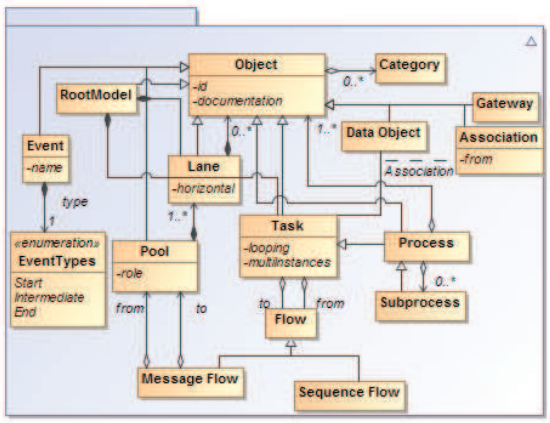


Fig. 6. Extract of the business process DSL

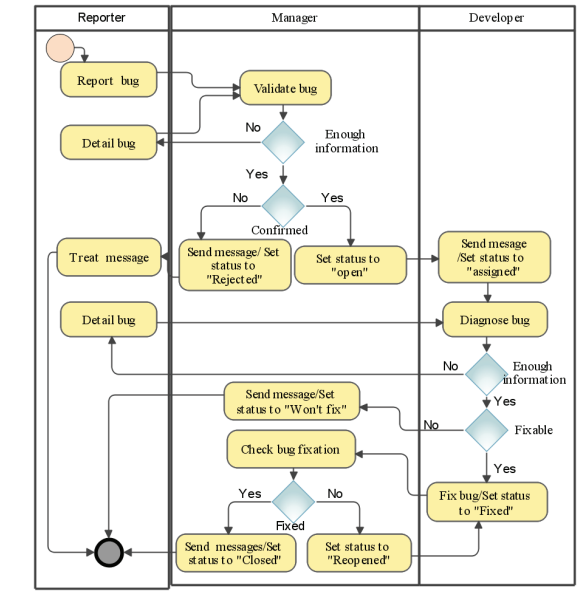


Fig. 7. Snapshot of BTS business process model

IV. Matching approach

In this section we briefly present our approach for establishing correspondences between heterogeneous models. It consists in analyzing input models in order to identify correspondences that exist among them and storing these correspondences into a model of correspondences. The elaboration of the correspondence model (the preliminary phase of the consistency management process) is briefly discussed below.

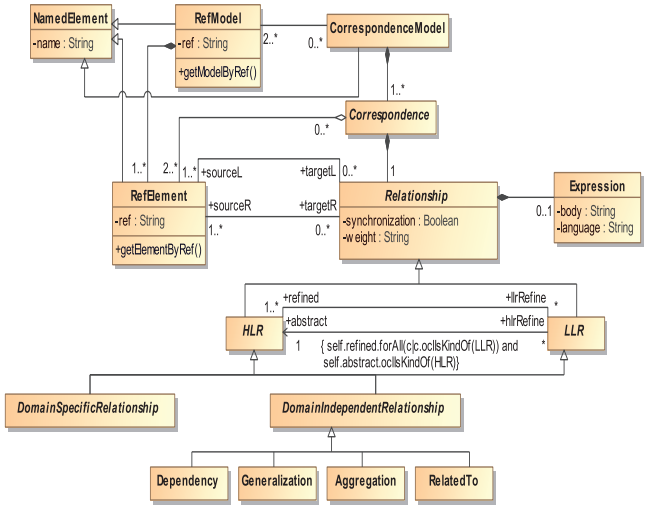
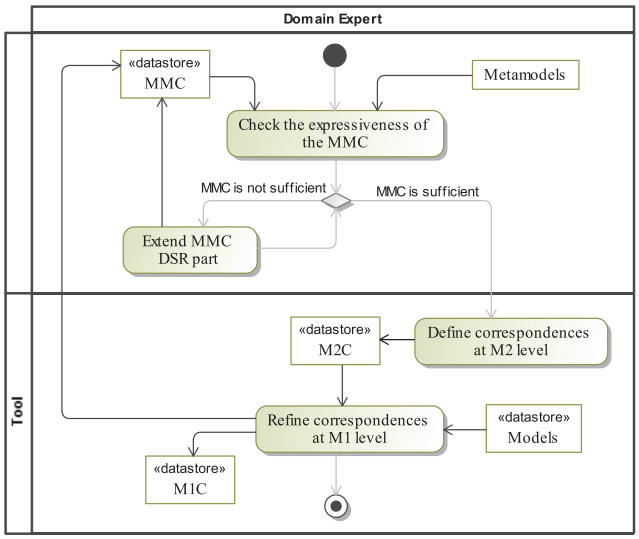


Fig. 8. Overview of the kernel part of the meta-model of correspondence (MMC)

In the context of heterogeneous matching, we have defined a Meta-Model of Correspondences (referred below as MMC). Figure 8 shows a kernel of MMC defining concepts and relations common to every application domain. The meaning of its main meta-classes is detailed below:

- CorrespondenceModel:a meta-class that represents all the correspondences established between at least two (meta)-elements belonging to different (meta)-models,
- Correspondence: in our work, we distinguish between a correspondence and a relationship. A correspondence is composed of a relationship and extremities which represent elements from input models,
- Relationship: an abstract meta-class that defines relationships between (meta-)elements of different (meta-)models. Linked to ”Element”, this meta-class allows, conceptually, defining n-ary correspondences connecting more than two elements at once. Its definition is done through specialization of ”Relationship”, by introducing two meta-classes: ”HLR” and ”LLR”,
- HLR (HighLevelRelationship): The specialization of this abstract meta-class will determine the relationships that will define correspondences at the meta-model level,
- LLR (LowLevelRelationship): The specialization of this abstract meta-class will determine the relationships that define correspondences at the model level,
- DomainIndependentRelationship: abstract meta-class that represents the generic relationships that may exist in different domains,
- DomainSpecificRelationship: an abstract meta-class representing relationships among models in specific domains. New relationships are specified by specialization of this meta-class according to the studied area.



MMC : Meta-model of correspondence
DSR: Domain Specific Relationship
M2C : Correspondence model between meta-models
M1C : Model of correspondence between models

Fig. 9. Activity diagram for the matching process

The correspondence model is constructed following the process described in Figure 9. Firstly, the process takes as input the various partial models, their respective meta-models and the meta-model of correspondences in its kernel part. Subsequently, a check is performed in order to inspect and ensure that the MMC contains enough relationships to set up correspondences among models, for a given application domain. If the domain expert considers that the proposed relationships are not sufficient to express correspondences that might exist between (meta-)model elements, the ”DomainSpecificRelationship” (DSR) meta-class of MMC is specialized. For example, to properly exploit the requirements model in the BTS domain, we must ensure that a given model element meets the requirement(s) to which it is linked. For that, the ”Verify” relationship is added as extension of DSR (Figure 10). It’s the same for ”UpdateValue”, used when a task of the business process model has to change the value of the element to which it is linked.

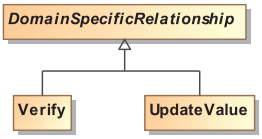


Fig. 10. Extract of the specific part of the MMC meta-model for BTS domain

Once the MMC contains the required relationships, the matching operation can be launched. It begins by identifying correspondences between meta-elements so as to produce the correspondences model called M2C. Correspondences stored in M2C are called High Level Correspondences (HLC).

Figure 11 shows examples of different types of HLCs. For example, the meta-element ”Requirement” on one side is linked to the meta-element ”Attribute” on the other side by a ”Verify” relationship. The meta-element ”Attribute” is also related to ”Task” through a ”Dependency” relationship. ”RelatedTo” is a ternary HLC defined between the meta-elements ”Task”, ”Entity” and ”Requirement”.

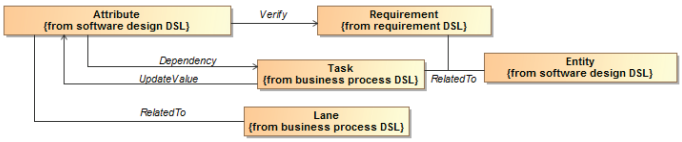


Fig. 11. Example of BTS HLCs

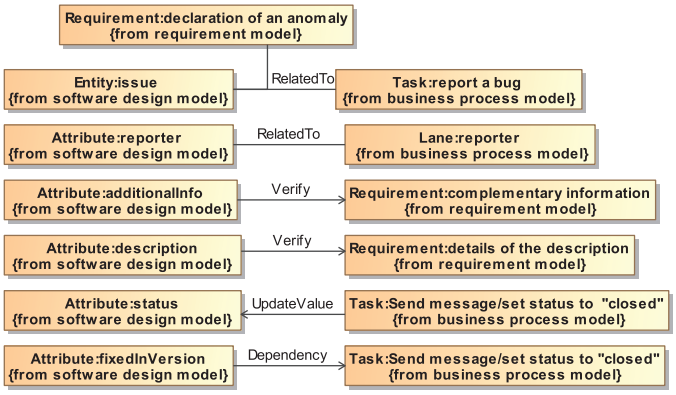


Fig. 12. Example of BTS LLCs

HLC are then refined in order to produce Low Level Correspondences (LLC) that link model elements. Please refer to [9] for more information about the refine concept that is no more described due to page limit.

Figure 12 describes LLCs created from the HLCs presented in Figure 11, by using the duplication ”refine” type. It’s possible to use the extension which is the second kind of ”refine” that extends the MMC meta-model by adding, if necessary, sub-classes of LLR (Figure 13). The model of correspondences will include the redefined correspondences (Figure 14).

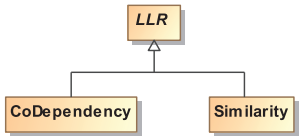


Fig. 13. Extension of LLCs

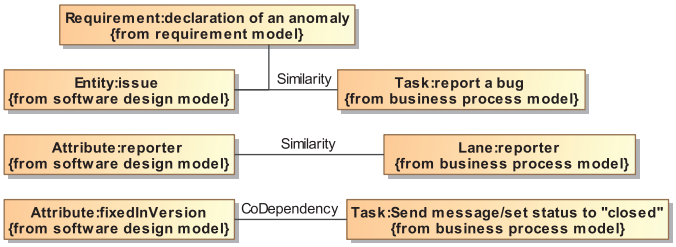


Fig. 14. Example of new LLCs

The proposed matching has several clear qualities. The first one is commonality: MMC provides a ”generic” part common to all domains - that defines a description of most common relationships. For so, we ran a survey on different DSLs belonging to different domains, to identify the most frequenty used ones. Variability is the second one. MMC can be extended depending on the specific aspects of the domain under consideration, in order to support the relationships relating to specific business areas in complex domains. This is done through specializations of the ”DomainSpecificRelationship” meta-class. The third one is flexibility. Thanks to it, the MMC can relate n models (through their model elements) and express n-ary cardinalities for each possible correspondence. Finally the correspondence model is lightweight. Indeed, the model of correspondence is built in a virtual manner [14] [15]. Elements are accessible through references and they do not have any physical existence in the model of correspondence.

The following section will discuss the consistency management based on the produced correspondence model.

V. Consistency Management Approach

In this section, we present the ”consistency management phase” which takes place after the matching phase. This phase is also a process, represented in Figure 15, that aims at describing the phases to perform after an evolution of source 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 description of the collaboration among these actors is not in the scope of this paper.

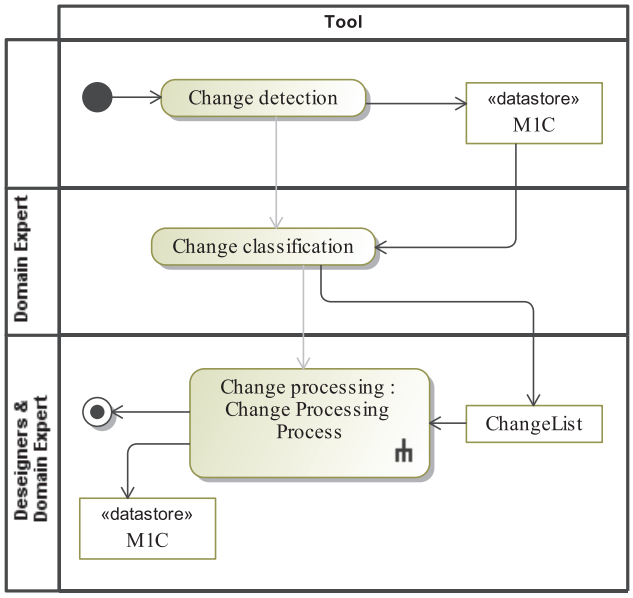


Fig. 15. Evolution management process

The process consists of three main phases which are: *change detection*, *change classification* and *change processing* (each of them will be described in the next subsections). The process takes as input the various models that may have evolved, and the model of correspondences (M1C). This latter is conform to the MMC meta-model and is obtained through the previous matching phase. Firstly, the change detection phase enables to trace changes that might have occurred on the various input models. These changes are stored in an extended part of the M1C (Figure 16). This is the fundamental step of the evolution management process because it allows to establish change processing once changes have been classified. Secondly a classification phase is performed. By involving the domain expert, it aims at classifying the changes stored in M1C and provides a change list. This is performed in order to manage impacts by assigning to each case a specific action. The final phase, called change processing, aims to ensure evolution of models by applying specific treatments on them. Some of these are done automatically whereas the others require the approbation of domain expert and designers. This process is iterative since the impact of changes may require an update of the correspondence model.

A. Change detection

1) Extension of MMC 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 MMC (see Figure 8) by adding a set of concepts, mimicking a CRUD [16] operation set, as described in the below, in order to take into account several types of change:

- History: meta-class used to keep a trace of the applied changes,
- DiffElt: Abstract meta-class that enables to store, once specialized, trace of the changed element,
- DeletedElt: Elements of models that no longer exist, as a result of a delete operation (If an element is deleted from a model, it will always be present in the M1C. So we need this meta-class to keep trace of the deleted ones),
- AddedElt: New model elements that was added to the initial models,
- ModifiedElt: New model element that is defined as a result of the modification of existing ones.

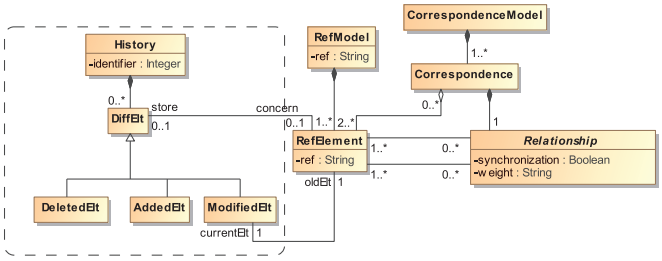


Fig. 16. Extract of the correspondence meta-model, oriented towards model evolution

Thanks to this extension, it is possible through the correspondence model to find, for each type of change on specific element the relationship it is connected to. Based on this, we can know the extremities elements.

2) Enrichment of the correspondence model: The extension of the meta-model presented in the previous sub-section, only allows to store the different changes without defining the way to do it. It is the purpose of this sub-section. Changes could be detected by means of model differencing techniques. As represented in Figure 17, in order to enrich the correspondence model with the different types of changes, we exploit the comparison engine EMFCompare [17]. EMFCompare is a Framework that provides a generic algorithm for calculating differences between two versions of a model, based on distance computing techniques.

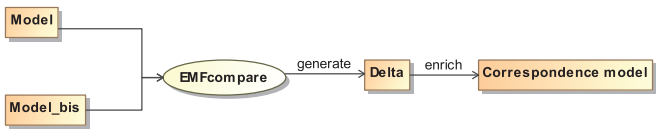


Fig. 17. Identifying changes using EMFCompare

But EMFComapare proves to be rather restrictive with two major drawbacks:

- A restriction on the number of models because comparison can be done with only two models of a given business domain,
- A mess of memory since the approach requires keeping in memory the previous version of the input model as well as the current one.

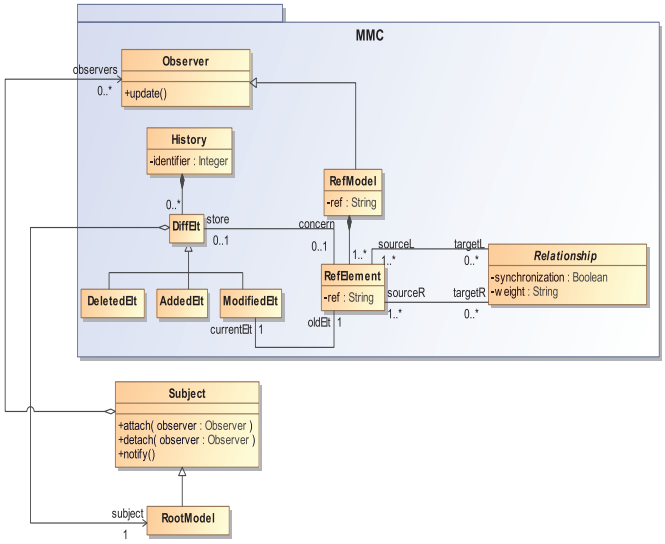


Fig. 18. Integration of the observer pattern in the MMC

Instead of parsing the whole model and checking if an element has changed, to overcome the mentioned problems, we gave up the use of an external tool. As presented in Figure 18, our solution is to use, an *observer design pattern* [18] [19] whose role is to inform the correspondence model by capturing, at real time, every change as a result of actions operated on model elements. The Figure shows the participants in the observer design pattern. It consists of a ”Subject” whose specialization (”RootModel”), called concrete subject, represents the observed element. This latter has methods to attach or detach an observer object and a method that sends notification to its observers whenever its state changes. The observers, through the update() method keep the state consistent with the concrete subject.

For clarity and simplicity reasons, we have decided to describe the M1C produced by the matching phase, using the extension ”refine” type, by the representation given in Table II (as presented in section 4). For example, the first line means that a requirement with ”id=1.1” is related to the model elements ”Report bug” and ”Issue” by a ”similaity” relationship.

Table III is a snapshot of some possible changes. For the modified type for instance, ”status” is changed to ”state” the ”fixedInVersion” element, which means the version where a bug was fixed, is replaced with ”targetMilestone”, which has another meaning (the version where a bug will be fixed). The analyst decides also that ”AdditionalInfo” has no use and that the description is enough, so he deletes it. The process engineer in his turn adds a new task called ”Set Status to reopend”.

TABLE II. SIMPLIFIED REPRESENTATION OF THE CORRESPONDENCE MODEL

Model element			Relationship
ReqM [1]	SDM[2]	BPM[3]	
1.1	”Issue”	”Report bug”	Similarity
1.1.3.1	”description”		Verify
1.1.3.2	”additionalInfo”		Verify
	”status”	”Set status to closed”	UpdateValue
	”fixedInVersion”	”Set status to closed”	CoDependency
	”reporter”	”reporter”	Similarity

[1] Requirement Model [2] Software Design Model [3] Business Process Model

TABLE III. SNAPSHOT OF THE POSSIBLE CHANGES

Type of change	Concerned Element
Modified	status → state, fixedInVersion → targetMilestone
Deleted	AdditionalInfo
Added	”Set Status to reopened”

B. Change classification

The classification of changes is used to manage impacts of changes by assigning to each type of change a specific 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 add a new model element, the matching process must be launched again,
- ”Monitored Evolution Category”: According to [20] when semantics comes into play, versioning problems become more complex to manage and cannot be performed automatically. In other words, automation support has typically to be reduced and user intervention is required to keep the desired degree of precision. For that this category includes actions that require a human assistance to decide about certain types of changes. For example, if one of the model elements 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.

The two first changes on Table III, concerning ”state” and ”fixedInVersion”, are classified into ”Monitored Evolution Category” whereas the other are classified into ”Automatic Evolution Category”.

C. Change processing

To maintain the consistency of the system with regard to established correspondences, model evolution must be performed. Figure 19 describes the process followed for the change processing. In this phase, models are updated to take into account the identified changes and the modifications deemed by the experts to be realized. On the one hand, the evolutions classified as ”Automatic Evolution Category” will be handled automatically through two strategies that correspond to the addition and the removal of model elements. For the first one, when a new model element is added to a source model, the matching phase is relaunched incrementally. The second strategy aims to delete a correspondence. In fact, if we delete a model element, the correspondence becomes orphan. We define an orphan correspondence as a correspondence for which one of its extremities is missing. When a correspondence is orphan it must be deleted from the correspondence model. On the other hand, evolutions classified as ”Monitored Evolution Category” imply a semi-automatic operation that offers evolution suggestions to guide the expert and help him to modify the model elements of the system. The correspondence is maintained if it is still valid after changing the model element. When an element is changed, it usually requires the modification of each element to which it is connected in regard to the established correspondences.

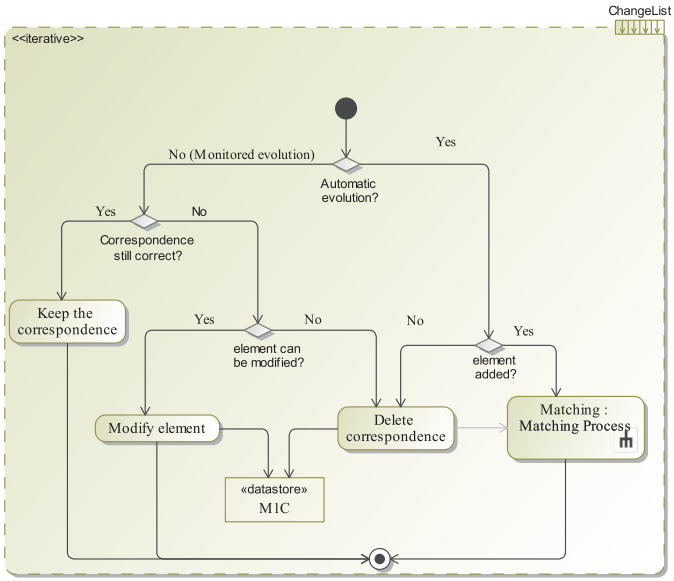


Fig. 19. Detailed vision of the change impact activity

Concerning the BTS example, firstly, as described in Table IV, the correspondence with ”CoDependency” relationship is deleted due to a semantic difference between the ”fixedInVersion” (new value) and ”targetMilstone” elements. On the contrary the correspondence related to the ”state” element will be kept even if it has been changed. Secondly, since ”additionalInfo” element has been deleted, the correspondence with the ”verify” relationship is deleted too and the related requirement element (with id=”1.1.3.1”) could be also deleted (it might not be needed anymore, depending on actors decisions). Finally, the matching phase is invoked to create a correspondence with an ”updateValue” relationship for the added element. The figure below describe the structure of the updated version of the correspondence model.

TABLE IV. SNAPSHOT OF M1C UPDATES

Model element			Relationship
ReqM [1]	SDM[2]	BPM[3]	
1.1	”Issue”	”Report bug”	Similarity
1.1.3.1	”description”		Verify
	”state”	”Set status to closed”	UpdateValue
	”state”	”Set status to reopened”	UpdateValue
	”reporter”	”reporter”	Similarity

[1] Requirement Model [2] Software Design Model [3] Business Process Model

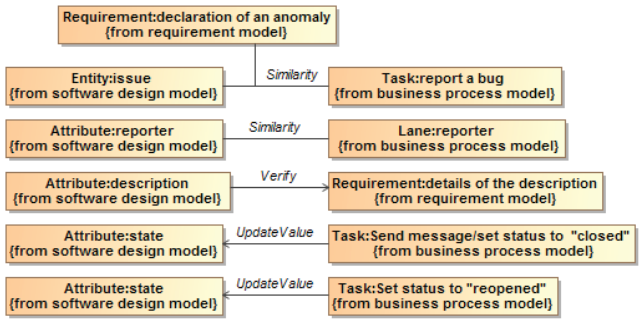


Fig. 20. Graphical representation of the new M1C

Since changes are managed and impacted to preserve models consistency, concurrent changes are not yet supported and will be treated in futur works.

VI. Tool support

To validate our approach, we have developed a matching tool suite called HMS (”Heterogeneous Matching Suite”). It is a suite of tools that gives stakeholders the ability to establish correspondences and manage the consistency between heterogeneous source models when they evolve.
For that, we decided to use Eclipse, the open source platform of development, considered as the main incubator of development projects [21] by the MDE community.

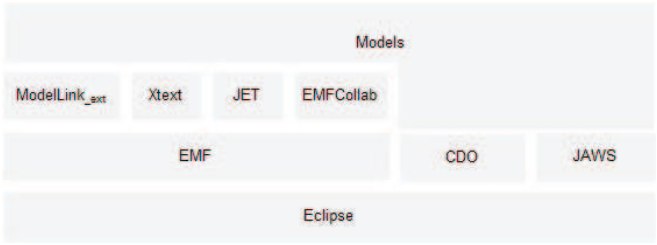


Fig. 21. Technical architecture of the HMS

The technical architecture that we adopted (Figure 21) is based on the following layers:

- EMF (Eclipse Modeling Framework) [22]: Component of the Eclipse MDT project and the basis of many MDE projects, EMF is an environment for modelling and code generation facility for building tools and other applications based on a structured data model,
- CDO (Connected Data Objects) [23]: CDO is a repository of models. This tool manages model persistence through a 3 layers architecture supporting EMF-based client applications, featuring a central model repository server and leveraging different types of pluggable data storage back-ends like relational databases, object databases and file systems,
- JAWS (Java API for WordNet Searching)[24]: an API that provides applications to retrieve data from the WordNet [25] database,
- JET (Java Emitter Templates) [22]: M2T transformation language based on EMF. it uses ”templates” to control the generated output in order to obtain the java code, XML ...
- EMFCollab [26]: a light-weight solution to let multiple users edit a single EMF model concurrently. EMFCollab has a client-server architecture. The server stores the master copy of the model. The server has the model loaded into memory all the time. On the clients side, EMFCollab stores a slave copy of the master model in the memory. This model is kept in sync over the network, by serializing and distributing the commands affecting the model. All clients see the same master copy over a single command stack,
- Xtext [27]: Xtext is a framework for development of textual domain specific languages. It covers all aspects of a complete language infrastructure, from parsers, over linker, compiler or interpreter,
- Modelink_ext [28]: Modelink is an EMF API for establishing graphical links between models using drag-an-drop. Modelink_ext is an extension of this latter making dynamic the number of model to link.

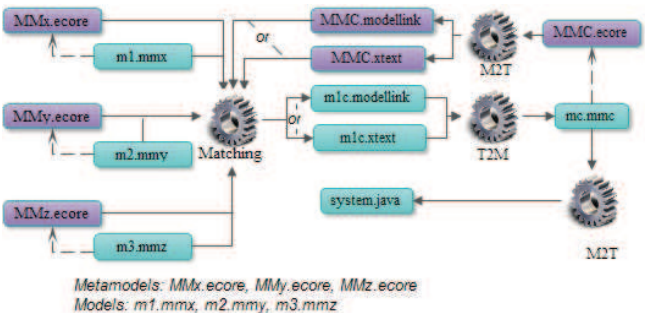


Fig. 22. Overall architecture of the prototype

The architecture of the matching phase of HMS prototype is described in Figure 22. It is composed of three modules represented by gears: Matching, M2T (Model To Text) and T2M (Text To Model). Rectangles represent the different (meta-)models in input and output of each module. To be suitable for different modes of work, the framework proposes two views: graphical and textual.

The graphical view is succinct and intuitive (using drag-and-drop). For this, we rely on Modelink_ext (Modelink that we improved by adding some extra functionalities). Regarding the textual view, it is suitable for stakeholders who prefer working directly on source mode editing. It must be noted that the two views are synchronized, meaning that a stockholder may start by exploiting the graphical view and continue on the same model of correspondences with the textual view and vice versa.

To be able to establish correspondences in a graphical manner, we must rewrite MMC to make it conform to the ModelLink syntax. Otherwise MMC must be described in Xtext syntax. To do this, the M2T module implements a Model to Text transformation which serializes - through JET technology - the meta-model of correspondences (MMC) into two kinds of models according to the chosen type of visualization. Once these models are available, correspondences between them can

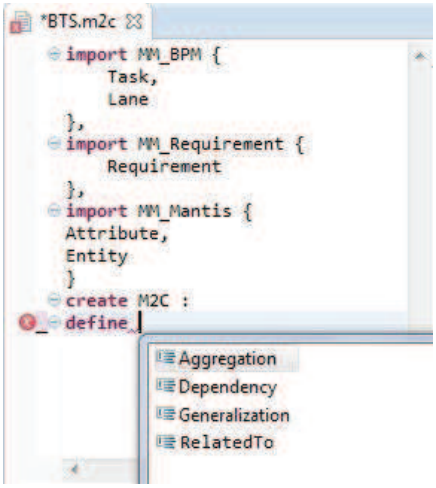


Fig. 23. Snapshot of the textual editor with the domaine independent relationships

be set up via the Matching module which is detailed below. The last module T2M parses the model of correspondences built using one of the visual tools (or both of them) into a model that conforms to MMC.

Figure 23 above illustrates the textual editor that allows creating the M2C (meta-level correspondence) model. In this editor we begin by referencing metamodel elements that will be connected by relationships. As shown on this figure, the editor displays, at the outset, only the relationships derived from the ”DomainIndependentRelationship” meta-class (”Aggregation”, ”Dependency”, ”Generalization” and ”RelatedTo”).

To define new relationships, the domain expert must extend the MMC metamodel. For this, a menu offers an ”Extend MMC” action. Figure 24 shows the addition of the ”Verify” relationship.

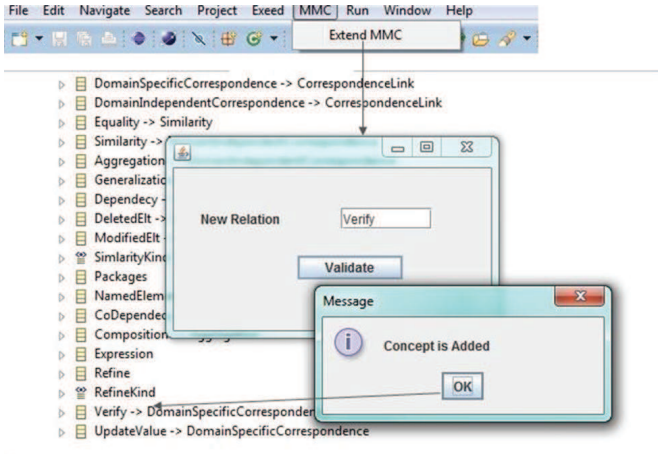


Fig. 24. Extension of the MMC

In the Figure 25, through the extension mechanism, the domain expert can begin to establish correspondences between the metamodels elements of the BTS. He is now able to use the ”Verify” as well as the ”UpdateValue” relationships (created the same way as above).

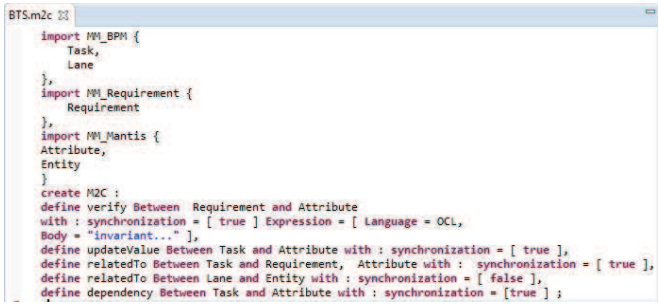


Fig. 25. Snapshot of the textual editor with new relationships

By performing a T2M (Text To Model) transformation, it is possible to have a tree view of the M2C model. The model M2C obtained (Figure 26) illustrates correspondences that may exist between the meta-models elements belonging to various business domains of the BTS. The correspondences previously defined are located below the node ”correspondence Model M2C”. For instance, the first correspondence that defines a ”verify” relationship relating an ”attribute” to a ”requirement”, has a synchronisation property set to ”true” and an expression written in OCL language.

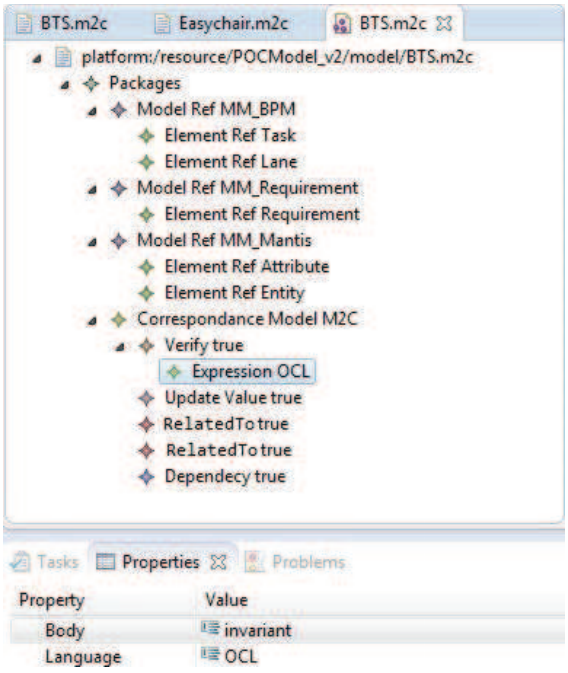


Fig. 26. Example of M2C for the BTS domain

Figure 27 shows an example of M1C produced by an extension refinement of type. The two new relationships ”CoDependency” and ”Similarity” are extensions of respectively ”Dependency” and ”RelatedTo” ones.

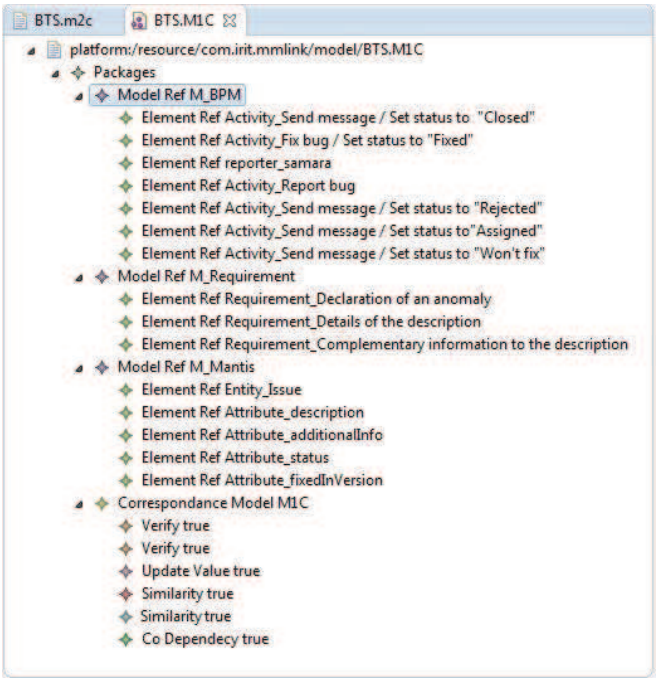


Fig. 27. Example of M1C for the BTS

The models have undergone different changes that are captured (see console of Figure 28) and exploited to enrich the model of correspondence. So far, the assisted tool for managing impacts has not been completed yet. The designers have to manually apply changes, based on the model of correspondences.

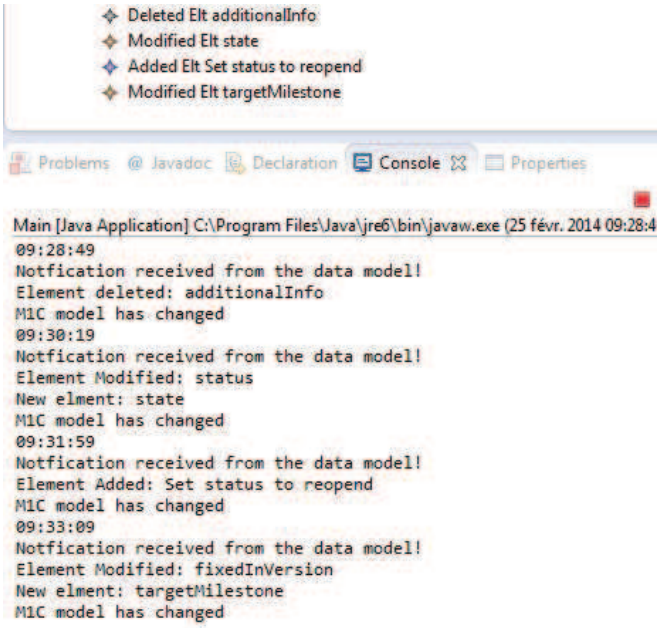


Fig. 28. Change detection and enrichement of the correspondence model

VII. Conclusion and perspectives

Our general research work addresses view-based complex information systems design. More precisely, our work aims to match heterogeneous source partial models related to a given system and to manage the consistency of those models when they evolve. Thereby, we are firstly interested in establishing correspondences between heterogeneous models described through different DSLs corresponding to different business areas of a domain. Secondly, models inevitably evolve during time. This brings out the need for a change management mechanism enabling the treatment of the impacts of changes on model elements and thereby ensuring the coherence of the system. This is done through a semi-automatic process that uses an extended correspondence model allowing to (i) detect changes made in a given input model, (ii) handle the modifications according to a 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. Using the Kuhne's terminology [29], a metamodel can provide both an ontological and a linguistic framework for model creation. A first perspective given to this work is to use domain ontologies or a least a linguistic resource such as Wordnet in order to alleviate the task of the expert by automating the creation of certain correspondences, predominately the Domain Independent Relationships. Secondly, we are working on taking into consideration the order of changes by exploiting weights and synchronization attributes of the relationship meta-class. Thirdly, we intend to complete the development of HMS prototype by adding a graphical syntax for matching, and providing the impact assisted tool. Finally, big industrial information systems include several designers working collaboratively. We have thus initiated a study to define a collaborative process to support the matching and evolution management activities. Indeed, in real complex systems, the designers should closely work together to efficiently produce the correspondence model, and to manage impacts due to models change.

References

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

[2] Z. Drey, C. Faucher, F. Fleurey, V. Mahé, and D. Vojtisek, "Kermeta language," *Reference Manual*, 2009.

[3] A. Zito, Z. Diskin, and J. Dingel, "Package merge in uml 2: Practice vs. theory?" *Model Driven Engineering Languages and Systems*, pp. 185–199, 2006.

[4] I. Ober, B. Coulette, and Y. Lakhrissi, "Behavioral Modelling and Composition of Object Slices Using Event Observation," in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS), Toulouse, 28/09/2008-03/10/2008*, ser. LNCS, J.-M. Bruel, K. Czarnecki, and I. Ober, Eds., no. 5301. http://www.springerlink.com: Springer, septembre 2008, pp. 219–233.

[5] A. Anwar, S. Ebersold, B. Coulette, M. Nassar, and A. Kriouile, "A rule-driven approach for composing viewpoint-oriented models," *Journal of Object Technology*, vol. 9, no. 2, pp. 89–114, 2010.

[6] M. El Hamlaoui, S. Ebersold, A. Anwar, M. Nassar, and B. Coulette, "Heterogeneous models matching for consistency management," in *ENASE 2013 - Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering, Angers, France, 4-6 July*, 2013, pp. 181–188.

[7] J. R. Williams, R. F. Paige, and F. A. Polack, "Searching for model migration strategies," in *Proceedings of the 6th International Workshop on Models and Evolution*. ACM, 2012, pp. 39–44.

[8] 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.

[9] 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*. ACM, 2011, pp. 30–38.

[10] 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*. IEEE, 2008, pp. 222–231.

[11] O. SysML, "Omg sysml-v1.1," http://www.sysml.org/docs/specs/OMGSysML-v1.1-08-11-01.pdf, November 2008. [Online]. Available: http://www.sysml.org/docs/specs/OMGSysML-v1.1-08-11-01.pdf

[12] mantisbt, "Mantis bug tracker," http://www.mantisbt.org/index.php.

[13] O. BPMN, "Omg bpmn-v2.0," http://www.omg.org/spec/BPMN/2.0/PDF, January 2011. [Online]. Available: http://www.omg.org/spec/BPMN/2.0/PDF

[14] C. Clasen, F. Jouault, and J. Cabot, "Virtualemf: a model virtualization tool," *Advances in Conceptual Modeling. Recent Developments and New Directions*, pp. 332–335, 2011.

[15] C. Clasen, F. Jouault, J. Cabot *et al.*, "Virtual Composition of EMF Models," in *7mes Journes sur l'Ingnierie Dirige par les Modles*, 2011.

[16] F. Barbier, S. Eveillard, K. Youbi, and E. Cariou, "Model-driven reverse engineering of cobol-based applications," *Information Systems Transformation. Architecture Driven Modernization Case Studies, Morgan Kauffman, Burlington, MA*, pp. 283–299, 2010.

[17] C. Brun and A. Pierantonio, "Model differences in the eclipse modeling framework," *UPGRADE, The European Journal for the Informatics Professional*, vol. 9, no. 2, pp. 29–34, 2008.

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

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

[20] A. Cicchetti and F. Ciccozzi, "Towards a novel model versioning approach based on the separation between linguistic and ontological aspects," in *ME 2013–Models and Evolution Workshop Proceedings*. CEUR-WS, 2013, p. 60.

[21] R. C. Gronback, *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*, 1st ed. Addison-Wesley Professional, 2009.

[22] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework*, 2nd ed. Addison-Wesley, 2009. [Online]. Available: http://my.safaribooksonline.com/9780321331885

[23] C. M. R. Overview, http://www.eclipse.org/cdo/. [Online]. Available: http://www.eclipse.org/cdo/

[24] J. A. for WordNet Searching, http://lyle.smu.edu/ tspell/jaws/. [Online]. Available: http://lyle.smu.edu/ tspell/jaws/

[25] L. Patil, M. Atique *et al.*, "A novel feature selection based on information gain using wordnet," in *Science and Information Conference (SAI), 2013*. IEEE, 2013, pp. 625–629.

[26] qgears, "emfcollab collaborative editing for emf models," http://qgears.com/products/emfcollab/, 2010.

[27] M. Eysholdt and H. Behrens, "Xtext: implement your language faster than the quick and dirty way," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. ACM, 2010, pp. 307–309.

[28] Modelink, http://www.eclipse.org/epsilon/doc/modelink. [Online]. Available: http://www.eclipse.org/epsilon/doc/modelink/

[29] P. Gómez, M. Sánchez, H. Florez, and J. Villalobos, "Co-creation of models and metamodels for enterprise architecture projects," in *Proceedings of the 2012 Extreme Modeling Workshop*. ACM, 2012, pp. 21–26.