Journal of Software Engineering

ISSN 1819-4311

Journal of Software Engineering 8 (3): 132-151, 2014
ISSN 1819-4311 / DOI: 10.3923/jse.2014.132.151
© 2014 Academic Journals Inc.

Towards a Framework for Heterogeneous Models Matching

[1,2]Mahmoud El Hamlaoui, [1]Sophie Ebersold, [3]Adil Anwar, [1]Bernard Coulette and [2]Mahmoud Nassar
[1]IRIT Laboratory, MACAO Team, University Toulouse 2-Le Mirail, Toulouse, France
[2]SIME Laboratory, IMS Team, University of Med V Souissi ENSIAS, Rabat, Morocco
[3]Siweb Laboratory, University of Med V Agdal, EMI, Rabat, Morocco

Corresponding Author: Mahmoud El Hamlaoui, IRIT Laboratory, MACAO Team, University Toulouse 2-Le Mirail, Toulouse, France

ABSTRACT
The overall goal of our approach is to relate models of a given domain that are created by different actors and thus are generally heterogeneous that is, described in different DSL (Domain Specific Languages). Instead of building a single global model, we propose to organize the different source models as a network of models which provides a global view of the system through a virtual global model. The matching of these models is done in a shared model of correspondences. We focus in this study on the elaboration of the model of correspondences, through a transformation called "refine". The approach is illustrated by a representative use case (a Bug Tracking System) and supported by a modeling tool called HMS (Heterogeneous Matching Suite).

Key words: Point of view, domain specific language, heterogeneity, correspondences, consistency

INTRODUCTION
Several approaches have been developed to face complex systems modelling. One of the most efficient and widely used in industry consists in elaborating separate models, called views that correspond to different points of view (Hilliard, 2001; Koning and van Vliet, 2006; Boulanger *et al*., 2010). This complex system modelling approach has long been used in software engineering and allows designers to focus on different parts of the system in isolation. If we consider the modelling of a new aircraft for instance, several designers build view-based models that express specific concerns: Mechanics, electricity, aerodynamics, software, etc. These view-based models should be related in order to represent the complex system. This linkage is one of the main issues to which designers must face.

Among problems that typically arise in this type of situation, we can mention the fact that different terminologies and terms may be used to represent the same concept or that the same term can be used to express different concepts. This issue is typically known as heterogeneity problem and designers of complex systems are facing hard problems because of it.

Heterogeneity has been initially tackled in various domains, namely: databases (Castano and De Antonellis, 2001), semantic web (Fenza *et al*., 2008), embedded systems (Eker *et al*., 2003), etc.

In MDE (Model Driven Engineering), models are described as first-class entities. A complex system can be represented as a set of separate, heterogeneous models (i.e. conform to different metamodels or expressed in different DSLs (Domain Specific Language) that tackle specific solutions in order to solve problems within specific contexts. The first solution that comes to mind is to compose those different source models into a global one, in order to have a single representation which is also easier to maintain. Our research team has been working for years on this composition issue by proposing a UML profile called VUML. However, when facing complex systems, this approach appears as too restrictive since source models are all UML models as described in Anwar *et al.* (2010) and Ober *et al.* (2008). Globally, composition approaches proposed in the literature (Drey *et al.*, 2009; Kolovos *et al.*, 2006a; Zito *et al.*, 2006) rely on the elaboration of one global model and have three major drawbacks related to models heterogeneity. The first disadvantage concerns the structure of the metamodel associated to the composed model; indeed, there is no consensus on how it should be constructed: from the union of all elements coming from the source models or from their intersection. The second disadvantage concerns the semantics used to represent a model element of a composed model given that the source models may use different semantics. Thirdly, a composed model is generally huge and in most cases impossible to maintain.

Instead of building a single global model, we propose a new approach consisting in organizing the different source models as a network of models that provides a global view of the system. This network is composed of models connected via relations called "correspondences". A correspondence links several model elements (two or more) with a specific semantic: equality, similarity, dependence, verification, etc. The set of correspondences are stored into a model called "Correspondence model". This model is produced by an expert designer, assisted by a tool. The expert designer first identifies high level correspondences among metamodels elements and then generates, assisted by our proposed tool, low level correspondences between models elements that refine high level ones.

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 and also strengthen the semantics of the models network. In other words, it significantly improves the quality of the representation link between the source models and the complex system.

Another interest of the proposed approach is to facilitate the maintenance of models. Indeed, as models tend to evolve, this may cause inconsistencies in the whole system. In fact, there is a need to identify and then reflect the changes, or at least to identify model elements that are impacted by them.

To sum up, the advantages of the proposed approach are:

- Produce a unique model of correspondences
- Maintain models consistency in case of evolution

In this study, we will focus on the first point as a continuation of our work (El Hamlaoui *et al.*, 2013a) in defining the refine concept and introducing the developed matching tool. The second point has been partially addressed in El Hamlaoui *et al.* (2013b).

ILLUSTRATIVE EXAMPLE

Objective and principle: To illustrate our approach, we have chosen an example based on a real project: BTS (Bug Tracking System). This system aims offer to different actors, based on their different roles (Team leader, developers, testers,...), the ability to report dysfunctions, comment them, track the status of an anomaly, notify collaborators of problems encountered, suggest solutions or possibilities of circumvention. The choice of this example is 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.

For this red line example, we consider that in the domain 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 metamodel and is manipulated by actors with specific roles (Fig. 1):

- **Requirement analyst:** Responsible for modelling end user needs (Business domain: User requirements). The produced model is expressed through a requirements DSL
- **Software architect:** Responsible for modelling anomalies (Business domain: Anomaly modelling). He creates a model expressed through a specific anomaly management DSL
- **Process engineer:** Responsible for bugs tracking process modelling (Business domain: Process modelling). He creates a model expressed through a business process DSL

In next subsections, we present the three models taken as examples, give extracts of their associated metamodels and show examples of correspondences between the models.

Requirement modelling: To assess the quality and validity of any project, one must ensure that it meets user requirements that are described by the requirement analyst. A requirement metamodel inspired from SysML notation (OMG, 2008) (Fig. 2) was chosen as DSL. 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 (Fig. 3) conform to the previous metamodel. For simplicity's sake, we limit the description of BTS to a few requirements. For instance, the requirement with id = "1.1" is related to the declaration of an anomaly; it includes a sub-requirement (id = "1.1.3") related to the summary of the anomaly, refined in its turn by additional constraints to be respected during the declaration of the anomaly.

Anomaly management modelling: In Fig. 4, we propose a software design DSL to define entities and associations between them. Based on this metamodel, we choose an open source software solution in the bug management field called Mantis (MantisBT, 2010) to represent the software design model. The Fig. 5 illustrates a snapshot of this model. The term "Issue" is used to

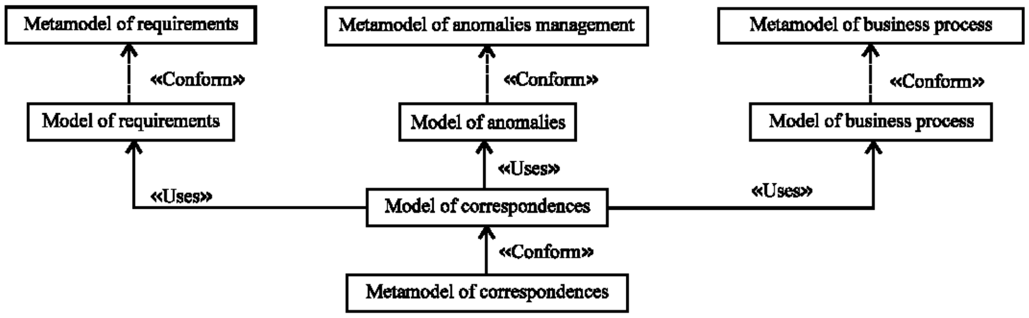

Metamodel of requirements
Metamodel of anomalies management
Metamodel of business process
«Conform»
«Conform»
«Conform»
Model of requirements
Model of anomalies
Model of business process
«Uses»
«Uses»
«Uses»
Model of correspondences
«Conform»
Metamodel of correspondences

Fig. 1: Global view of the BTS models and metamodels

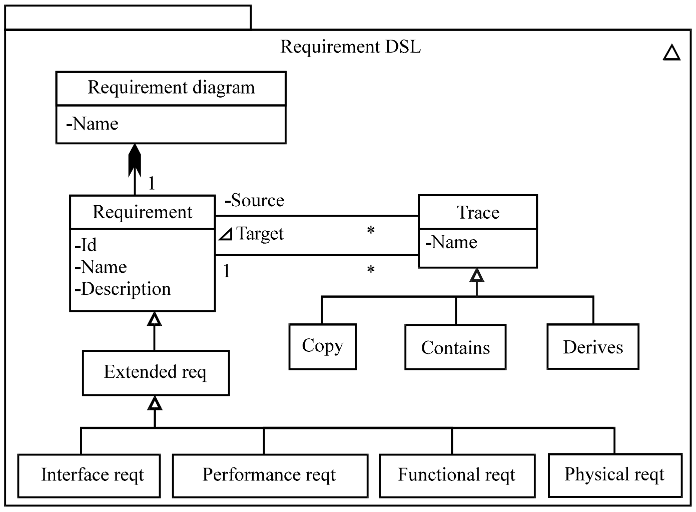


Fig. 2: Extract of the requirement metamodel

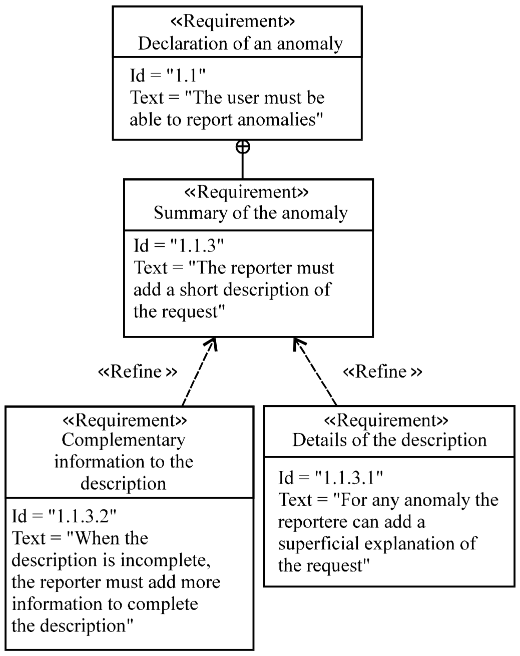

«Requirement»
Declaration of an anomaly
Id = "1.1"
Text = "The user must be able to report anomalies"
«Requirement»
Summary of the anomaly
Id = "1.1.3"
Text = "The reporter must add a short description of the request"
«Refine»
«Refine»
«Requirement»
Complementary information to the description
Id = "1.1.3.2"
Text = "When the description is incomplete, the reporter must add more information to complete the description"
«Requirement»
Details of the description
Id = "1.1.3.1"
Text = "For any anomaly the reportere can add a superficial explanation of the request"

Fig. 3: Snapshot of the BTS requirement model

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 (steps to reproduce) and two types of involved persons with the following roles: “reporter” and ”assigned To”. 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.

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. We suppose that the process

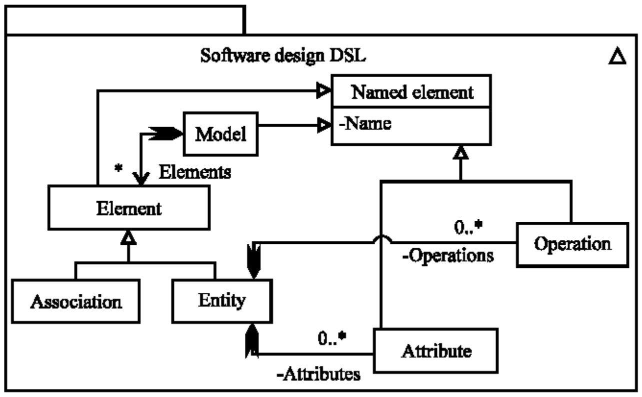


Fig. 4: Extract of the software design metamodel

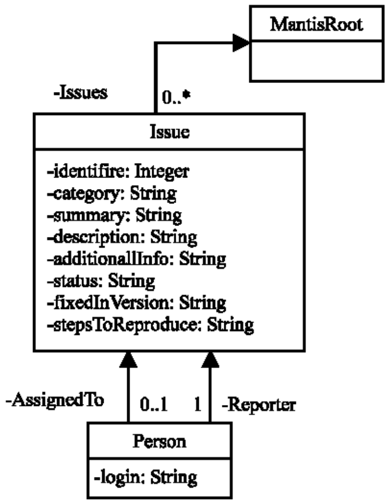


Fig. 5: Snapshot of the software design model

engineer used BPMN (OMG, 2011a) for modelling the business process. The BPM metamodel (Fig. 6) comprises the following concepts: “lane”, “pool”, “flow”, “process”, etc. A snapshot of the process expressed in conformity with BPM is presented in Fig. 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 to “Fixed”. The process manager, notified by the change, rechecks the proposed solution and modifies the anomaly status to “closed”, if it has been successfully corrected.

Exemple of BTS correspondences: Figure 8 exhibits a set of correspondences identified by the design expert between the three models of BTS domain presented above. They are based on

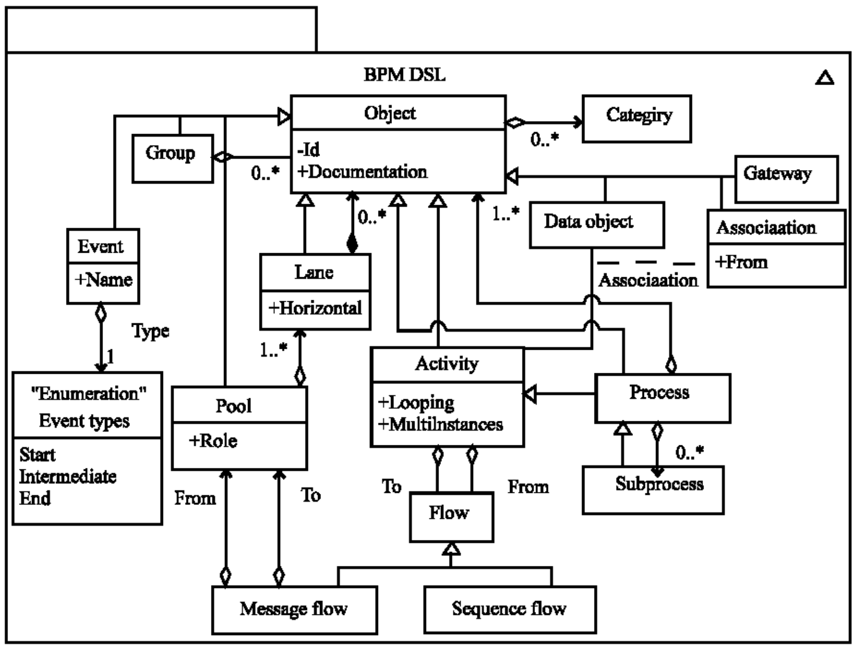


Fig. 6: Extract of the BPM metamodel

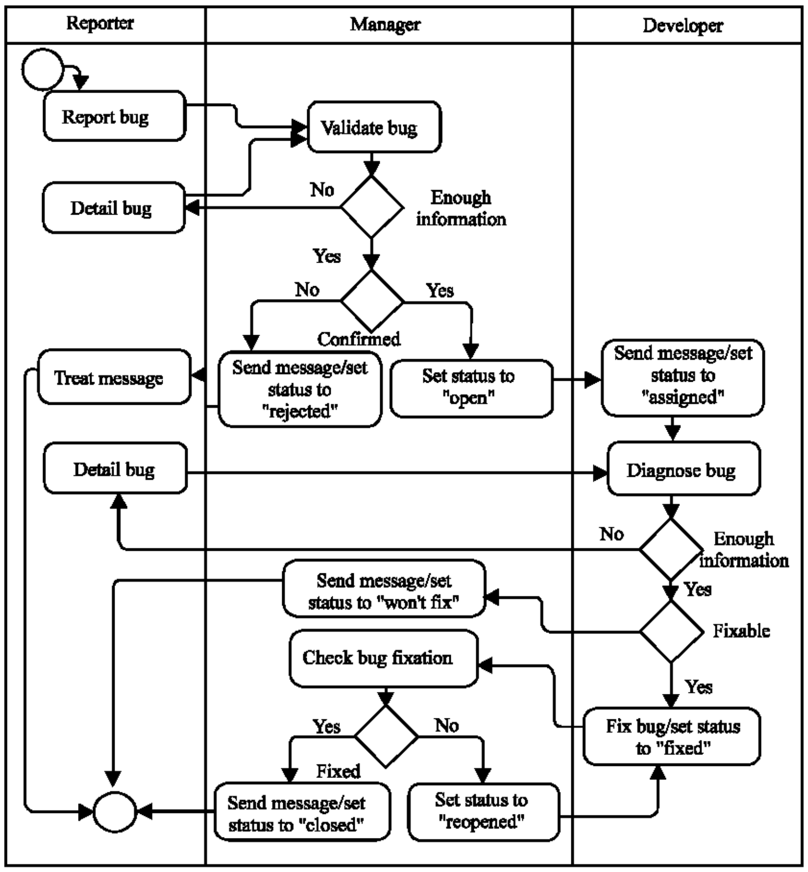


Fig. 7: Snapshot of the BTS business process model

"Equality", "Similarity", "CoDependency", "Verify" and "UpdateValue" relationships. For example, the elements "reporter" being the same in both software design and business process models, they have been related through an "Equality" relationship. The task "Set status to reopened" of the business process model and the attribute "status" of the software design model have been related through an "Update value' relationship because the value of the attribute "status" must be updated. A 3-ary "Similarity" relationship has been put among elements of the three models. All the identified correspondences are contained in a model of correspondence.

MATCHING APPROACH

Here, we present our approach for establishing correspondences between heterogeneous models. It consists in analysing input models in order to identify correspondences that exist among them and storing these correspondences into a model of correspondences. We discuss below the elaboration of the correspondence model as well as the proposed matching process.

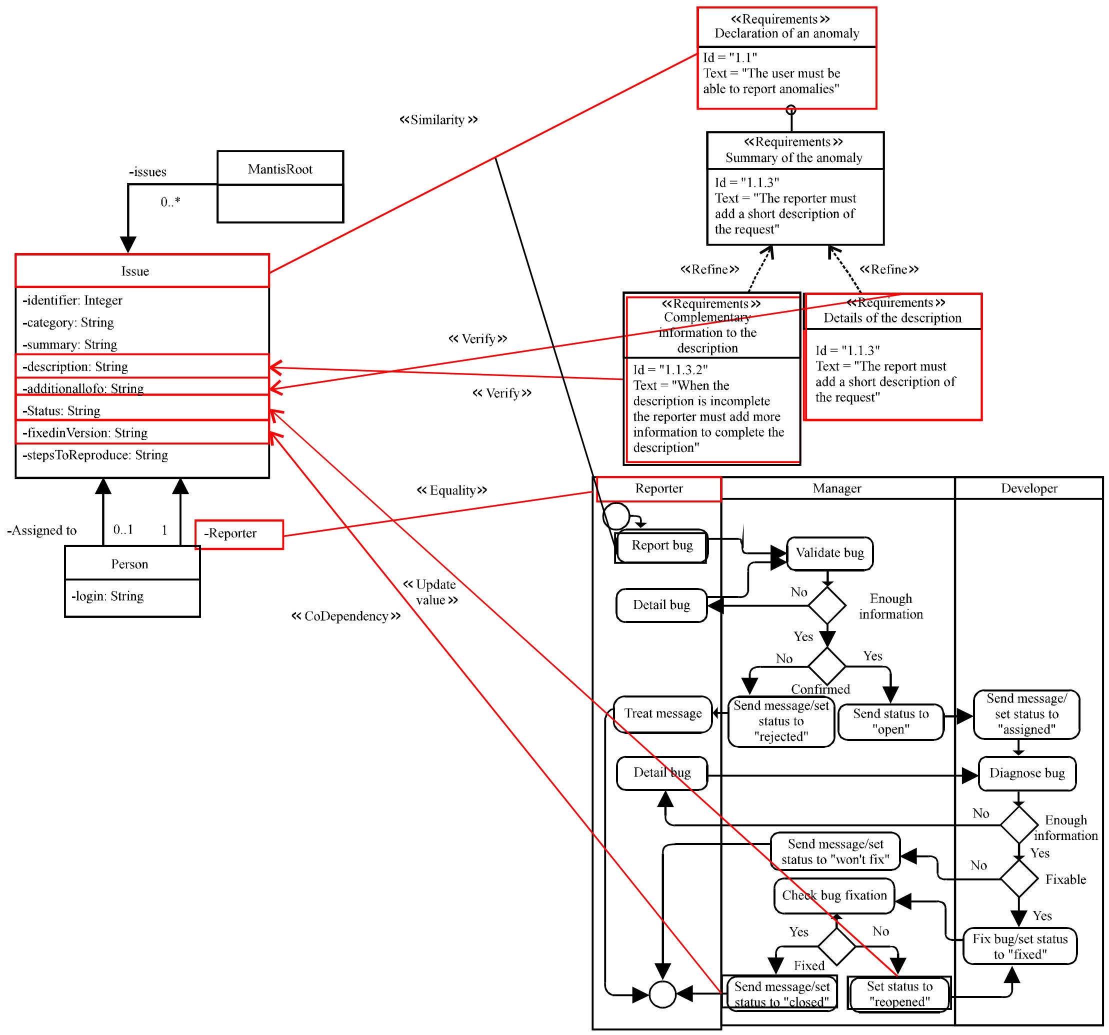

«Requirements» Declaration of an anomaly
Id = "1.1" Text = "The user must be able to report anomalies"
«Similarity»
«Requirements» Summary of the anomaly
Id = "1.1.3" Text = "The reporter must add a short description of the request"
-issues
0..*
MantisRoot
«Refine»
«Refine»
Issue
-identifier: Integer
-category: String
-summary: String
-description: String
-additionalIofo: String
-Status: String
-fixedinVersion: String
-stepsToReproduce: String
«Verify»
«Verify»
«Requirements» Complementary information to the description
Id = "1.1.3.2" Text = "When the description is incomplete the reporter must add more information to complete the description"
«Requirements» Details of the description
Id = "1.1.3" Text = "The report must add a short description of the request"
«Equality»
-Assigned to
0..1
1
-Reporter
Person
-login: String
«Update value»
«CoDependency»
Reporter
Manager
Developer
Report bug
Validate bug
Detail bug
No
Enough information
Yes
No
Yes
Confirmed
Treat message
Send message/set status to "rejected"
Send status to "open"
Send message/ set status to "assigned"
Detail bug
Diagnose bug
No
Enough information
Yes
Send message/set status to "won't fix"
No
Fixable
Check bug fixation
Yes
Yes
No
Fix bug/set status to "fixed"
Fixed
Send message/set status to "closed"
Set status to "reopened"

Fig. 8: Examples of correspondences among BTS source models

Correspondence metamodel: In the context of heterogeneous matching, we have defined a metamodel of correspondences called "MMC" in the following Fig. 9. The meaning of its main meta-classes is detailed in the following paragraph:

- **Correspondence model:** A meta-class that represents all the correspondences established between at least two (meta)-elements belonging to different (meta)-models
- **Correspondence:** In our study, 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: "Domain independent relationship" and "domain specific relationship"
- **Domain independent relationship:** Abstract meta-class that represents the generic relationships that may exist in different domains
- **Domain specific relationship:** 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

Matching process: The correspondences model cannot be constructed in a monolithic manner. It must be created following a given process called "matching process". The proposed matching process, Fig. 10, aims at describing the steps required to perform the matching among heterogeneous source models, in order to obtain a correspondences model. The produced model is called M1C (model of correspondence at M1 level) in the following and contains the correspondences between elements of models representing the studied domain. It involves two actors, namely, a domain (design) expert who can be seen as an orchestrator of the system and a tool to support the automated phases.

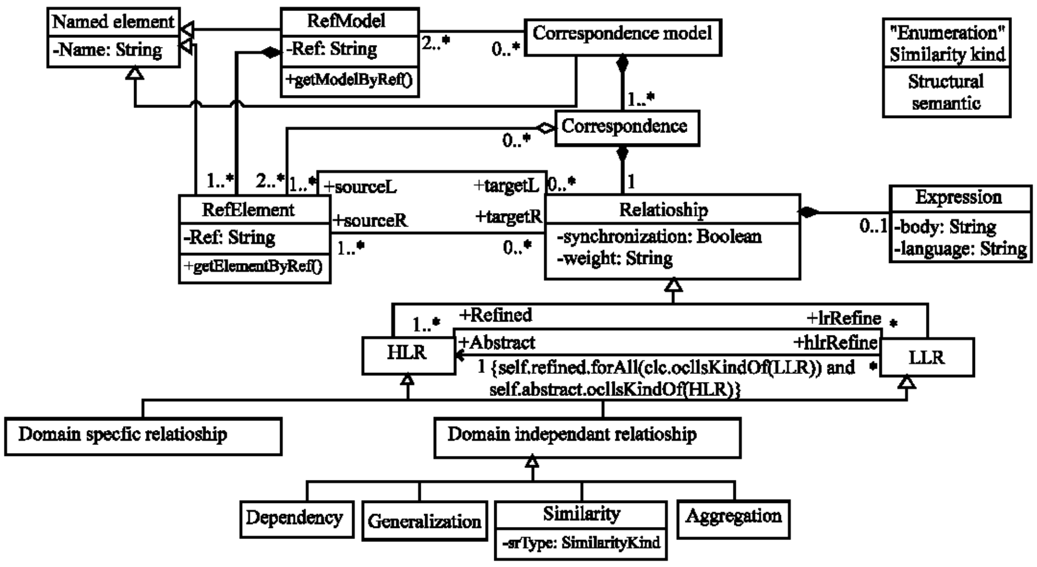


Fig. 9: Overview of the generic part of the metamodel of correspondences

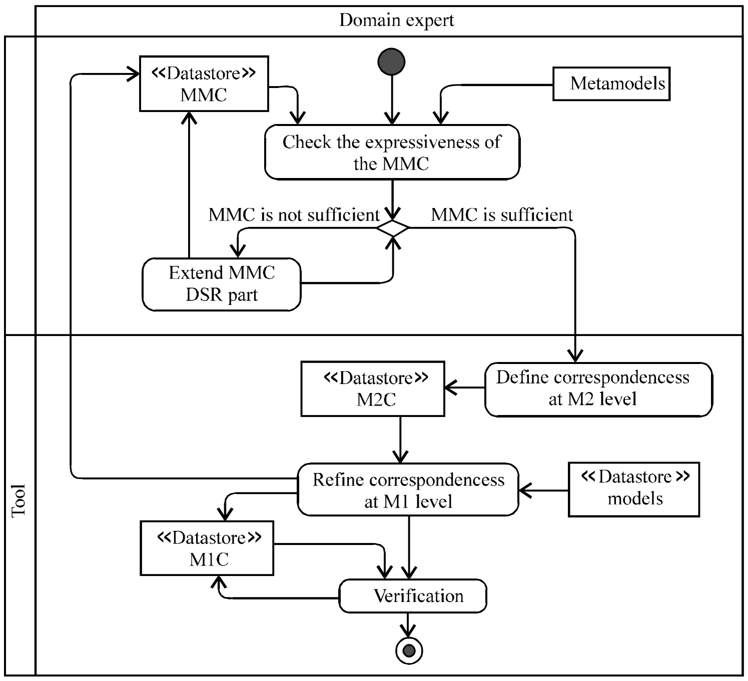


Fig. 10: Activity diagram for the matching process, MMC: Meta-model of correspondence, DSR: Domain specific relationship, M2C: Correspondence model between meta-models, M1C: Model of correspondence between model

Firstly, the process takes as input the various models, their respective Metamodels and the Metamodel of Correspondences (MMC) in its generic 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, we use a requirements model in our BTS domain, we must ensure that a given (meta-)model element meets the requirement(s) to which it is linked. For that, the "Verify" relationship is added as extension of DSR.

Once the MMC contains the necessary 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). HLC are thus refined in order to produce Low Level Correspondences (LLC) that link models elements, through a process that is described further. This refine extends the MMC by adding, if necessary, sub-classes of LLR and creates the final model of correspondences called M1C which contains the correspondences between model elements. Verification is the last step of this process. It consists in checking that refinements have been correctly done. To that end, the domain expert must ensure that the model elements are properly related based on his knowledge of the domain and the semantics of the relationships.

REFINEMENT OF HIGH LEVEL CORRESPONDENCES

As explained in the matching process presented the study proposed in approach to first specify correspondences at the abstract level (M2) in order to minimize the modelling effort and then to reuse them through refine transformation at the concrete level (M1).

Refine concept: Refinement is a classical way to reuse in software and system engineering (Wolf, 1994; Wooldridge, 1997; Medvidovic and Taylor, 2000). It can be seen as crossing different levels of abstraction with the purpose of adding details when passing from a given level to a more concrete one.

In the context of Model Driven Architecture (MDA) (OMG, 2003) that notion may be represented as a transformation of a Platform Independent Model (PIM) that exists at a high level of abstraction to a Platform Specific Model (PSM) that exists at a lower one. Most elements from the abstract model (PIM) are copied into the refined model (PSM) while other elements must be changed in order to ensure specific properties.

According to Wagelaar (2005), even though refinement is a key concept in MDA, it is loosely defined and open to misinterpretation.

The refine notion has also been defined in UML (2007) as a stereotype for “Abstraction”. Abstraction is a directed relation from an element to another one stating that the dependent element (concrete) depends on the other one (abstract):

The study distinguish two types of correspondences (Fig. 11):

- **Correspondence between metamodel elements:** “High Level Correspondence” that is called HLC. It connects meta-elements via., a High Level Relationship (HLR)
- **Correspondence between model elements:** “Low Level Correspondence” that are called LLC. It connects elements via., a Low Level Relationship (LLR)

A transition from HLC to LLC is similar to a transformation of a PIM into a PSM in the context of MDA. This is done by projecting abstract correspondences on the concrete level.

Starting by identifying HLR between meta-elements at the metamodel level (M2C) allows establishing, in a second step, relationships between elements at the model level (M1C). The principle consists in defining a correspondence once at the metamodel level and then reuses it each

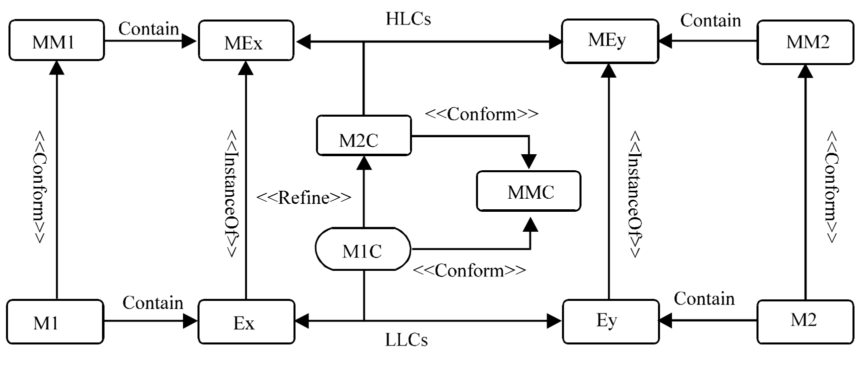


Fig. 11: Overview of the transition between M2C and M1C, Metamodels: MM1, MM2, Meta-elements: MEx, Mey, Models: M1, M2, elements: Ex, Ey

time it is needed at the model level. In other words, correspondences among metamodel elements induce correspondences between model elements. Cx = [Similarity (Entity, Requirement)], Cy = [Similarity (Issue, Declaration_of anomaly)], respectively examples of HLC and LLC.

The proposed "refine" is a transformation between elements of different abstraction levels, more specifically between HLC and LLC. It is possible by the refine, to complete, if necessary, the description of the HLC with some required functionalities. A HLC allows to anticipate the complexity of the matching (due to the establishment of correspondences directly at the model level), by first establishing, correspondences between metamodels. Thereafter, an accuracy of certain details of the abstract model can be managed at the LLC level, obtained by refine transformation of HLC.

Figure 11 summarizes the link between MMC, M2C and M1C and the level of their use. The study presented a refine (denoted by R_f) is constructed as follows.

Ci R_f Cj, with Ci and Cj as two correspondences, iff Ci is defined from Cj which means that Ci is an upgrading of Cj by adding details in order to precise the correspondence.

High level correspondences creation: Metamodels are the cornerstone of our approach since correspondences at the metamodel level (HLC) induce correspondences at the model one (LLC).

The definition of HLCs is done only once during the modelling cycle but they are exploited to generate LLCs. In other words, the M2C model which is conform to the MMC, is used as input to establish LLCs (stored in the M1C model).

So far the study revealed that identification of correspondences is done manually by the domain expert. He is supposed to know the relationships that may exist between the meta-elements and their meaning. Nevertheless, we have started to partially automate this task by using linguistic resources. The result of this work is not addressed in this study.

Figure 12 shows examples of three different types of correspondence. The first one relates the meta-element "Requirement" on one side to the meta-element "Attribute" on the other side by a "Verify" relationship that must be added first as a sub-class of "domain specific relationship" in the MMC,. The meta-element "Attribute" is also related to Task through a "Dependency" relationship. Thirdly, a "Similarity" is defined between the meta-elements "Task" and "Requirement".

A correspondence at the meta-level cannot be instantiated as such at the model level. It is necessary, depending on needs, to enrich the relationship of a correspondence to adapt it at the model level.

Low level correspondences creation: We present below the process that shows how LLCs are built. First, one must identify elements to relate. After that, correspondences are created are performed according to two scenarios: Duplication and Extension.

Duplication: This type of refine is a homomorphism-a structural preservation from one algebraic structure to another- between correspondences in M2C and M1C levels. Its role is to duplicate all

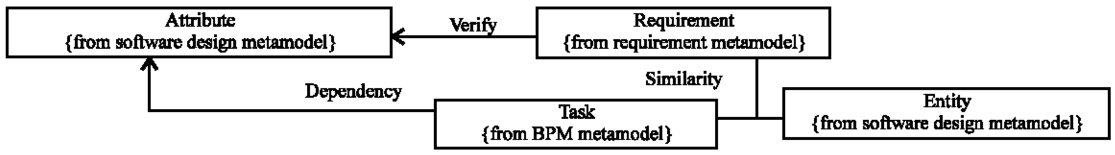

Attribute
{from software design metamodel}
Verify
Requirement
{from requirement metamodel}
Similarity
Dependency
Task
{from BPM metamodel}
Entity
{from software design metamodel}

Fig. 12: Example of BTS HLCs

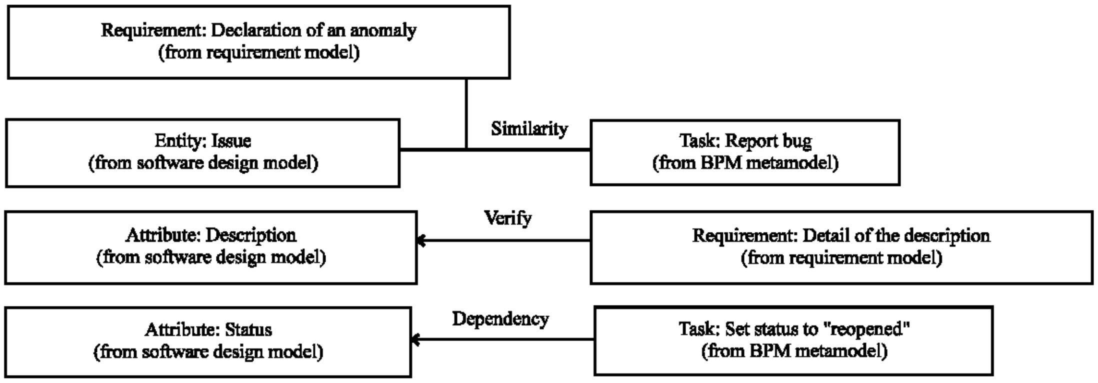

Requirement: Declaration of an anomaly (from requirement model)
Similarity
Entity: Issue (from software design model)
Task: Report bug (from BPM metamodel)
Verify
Attribute: Description (from software design model)
Requirement: Detail of the description (from requirement model)
Dependency
Attribute: Status (from software design model)
Task: Set status to "reopened" (from BPM metamodel)

Fig. 13: Example of BTS LLCs

the relationships of the correspondences defined at the meta-level (between meta-elements) at the model level (between model elements).

Figure 13 describes some LLCs created by refining HLCs presented in Fig. 12.

Extension: This type of refine aims to redefine the correspondences. A HLC can be reformulated to add constraints, exceptions and/or specific treatments.

LLCs created by the duplication scenario, may not be totally suitable for the expert designer. He may have to make choices about certain actions to be performed (Cariou *et al.*, 2009) to preserve the desirable properties for example or to add details or information on correspondences, so as to precise the semantics.

This redefinition involves the creation of a new correspondence and a new meta-class in MMC (sub-class of LLC in the MMC).

In the BTS example, the domain expert realized that the "dependency" relationship (Fig. 13) relating two model elements is semantically too weak and insufficient. It should be refined into a "co-dependency" relationship. This relationship defines a mutual dependency between model elements, where any change concerning one of them may affect the others. The same thing could happen, according to the needs, with "similarity" and "aggregation" relationships which could be refined, respectively as an "equality" and a "composition relationship". The first one is used when one relates identical model elements, i.e. having the same structural and semantic descriptions, for instance a model element duplicated in several models. The second one is used to express a strict form of "aggregation", where the life cycle of parts depends on the whole.

TOOL SUPPORT

To validate our approach, we are developing a matching tool called HMS ("Heterogeneous Matching Suite"). It is a suite of tools that gives stakeholders the ability to establish correspondences between heterogeneous models.

Fort hat, we decided to use Eclipse, the open source platform of development, considered as the main incubator of development projects (Gronback, 2009) by the MDE community.

Functional architecture OF HMS: The architecture of HMS prototype is described in Fig. 14. 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 (Epsilon, 2010) which 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 graphical manner, we must rewrite MMC to make it conform to the ModelLink syntax. Otherwise MMC must be described in Xtext (Eysholdt and Behrens, 2010) syntax.

To do this, the M2T module implements a Model to Text transformation which serializes-through JET (Java Emitter Template) technology (Steinberg *et al.*, 2009) the correspondence metamodel (MMC) into two kinds of models according to the chosen type of visualization.

Once these models are available, correspondences between them can 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.

Application to the BTS example: Figure 15 below illustrates the textual editor that allows creating the M2C 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 “domain independent relationship” meta-class (“Aggregation”, “Dependency”, “Generalization” and “similarity”).

To define new relationships, the domain expert must extend the MMC metamodel. For this, a menu offers an “Extend MMC” action. Figure 16 shows the addition of the “Verify” relationship.

In the Fig. 17, 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).

By performing a T2M (Text To Model) transformation, it is possible to have a tree view of the M2C model. The model M2C obtained (Fig. 18) illustrate correspondences that may exist between the metamodels 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

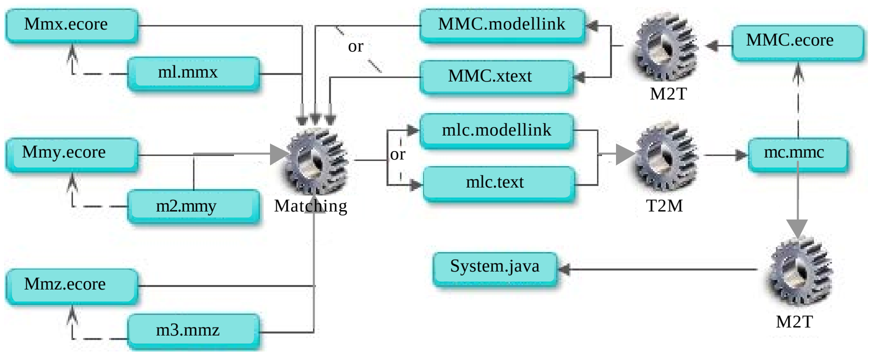
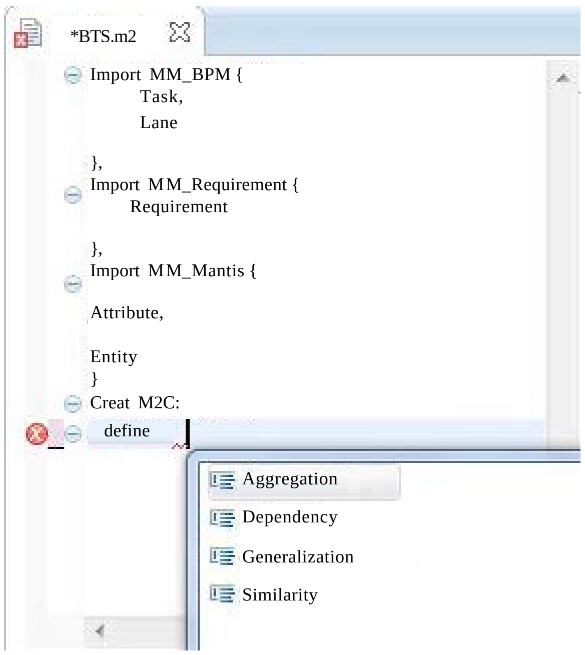


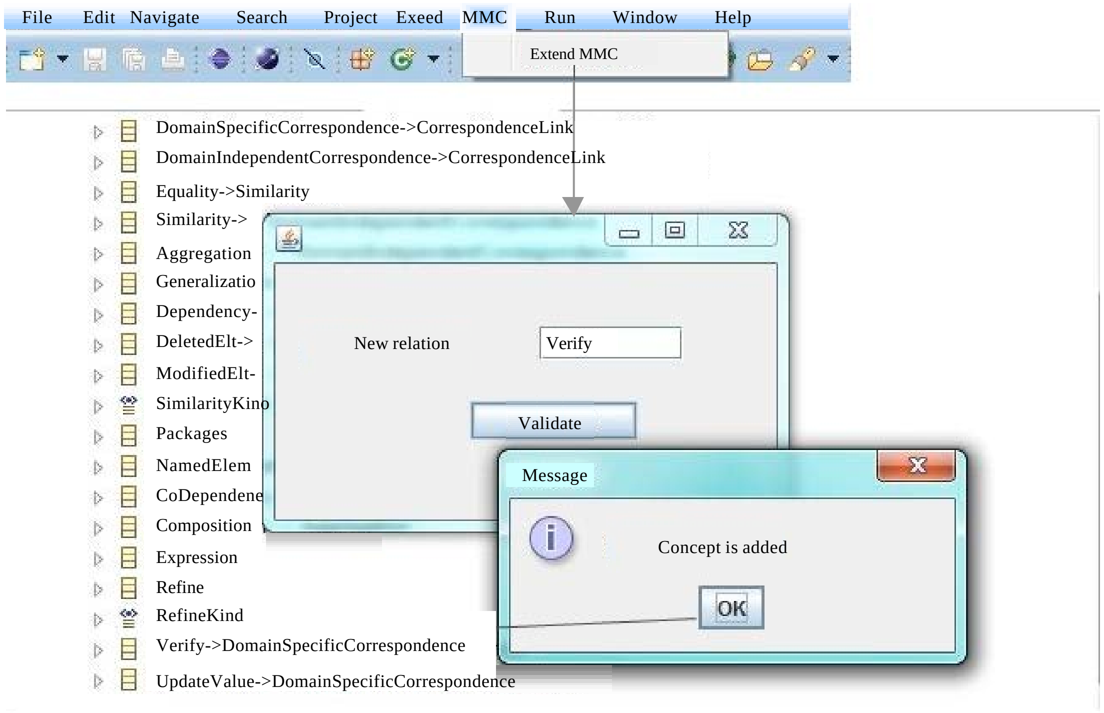
Fig. 14: Overall architecture of the prototype, Metamodels: Mmx.ecore, Mmy.ecore, Mmz.ecore, Models: m1.mmx.m2.mmy.m3.mmz



*BTS.m2

```
Import MM_BPM {
        Task,
        Lane
},
Import MM_Requirement {
        Requirement
},
Import MM_Mantis {
Attribute,
Entity
}
Creat M2C:
 define
```

Fig. 15: Snapshot of the textual editor with DIR



DomainSpecificCorrespondence->CorrespondenceLink
DomainIndependentCorrespondence->CorrespondenceLink
Equality->Similarity
Similarity->
Aggregation
Generalizatio
Dependency-
DeletedElt->
ModifiedElt-
SimilarityKino
Packages
NamedElem
CoDependene
Composition
Expression
Refine
RefineKind
Verify->DomainSpecificCorrespondence
UpdateValue->DomainSpecificCorrespondence

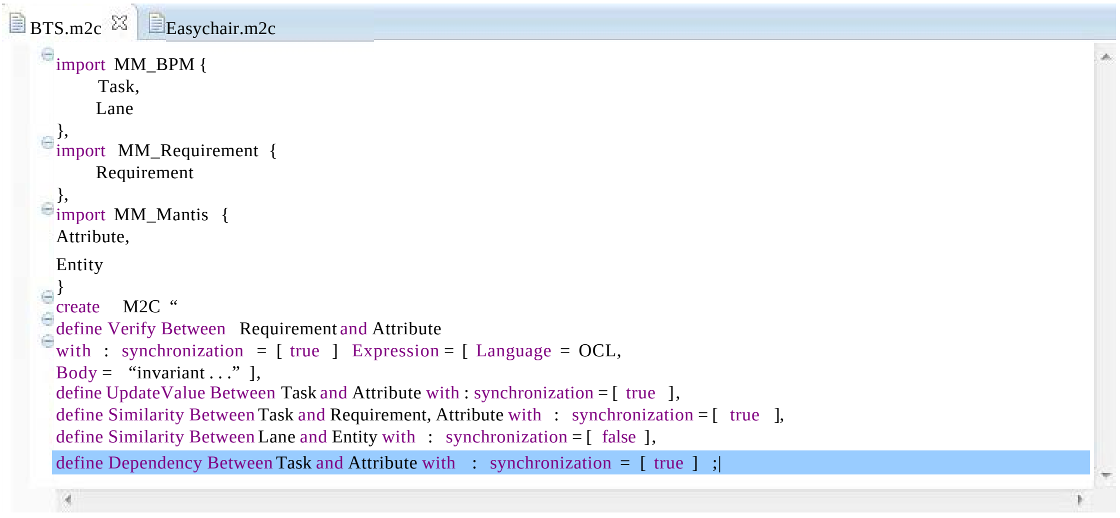
New relation: Verify

Message

Concept is added

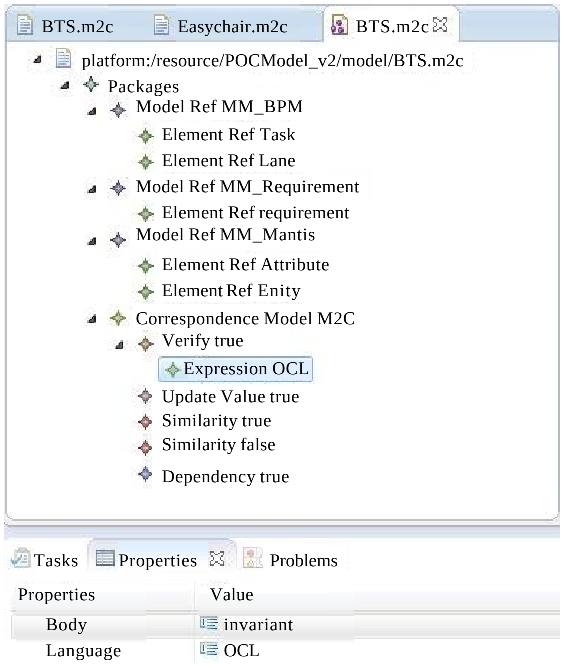
Fig. 16: Extension of the MMC

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.



```
import  MM_BPM {
        Task,
        Lane
},
import  MM_Requirement  {
        Requirement
},
import  MM_Mantis   {
Attribute,
Entity
}
create    M2C  “
define Verify Between   Requirement and Attribute
with   :  synchronization   =  [  true   ]   Expression = [  Language  =  OCL,
Body =    “invariant . . .”  ],
define UpdateValue Between Task and Attribute with : synchronization = [  true   ],
define Similarity Between Task and Requirement, Attribute with   :   synchronization = [   true   ],
define Similarity Between Lane and Entity with   :  synchronization = [  false  ],
define Dependency Between Task and Attribute with   :   synchronization  =  [  true  ]   ;|
```

Fig. 17: Snapshot of the textual editor with new relationships



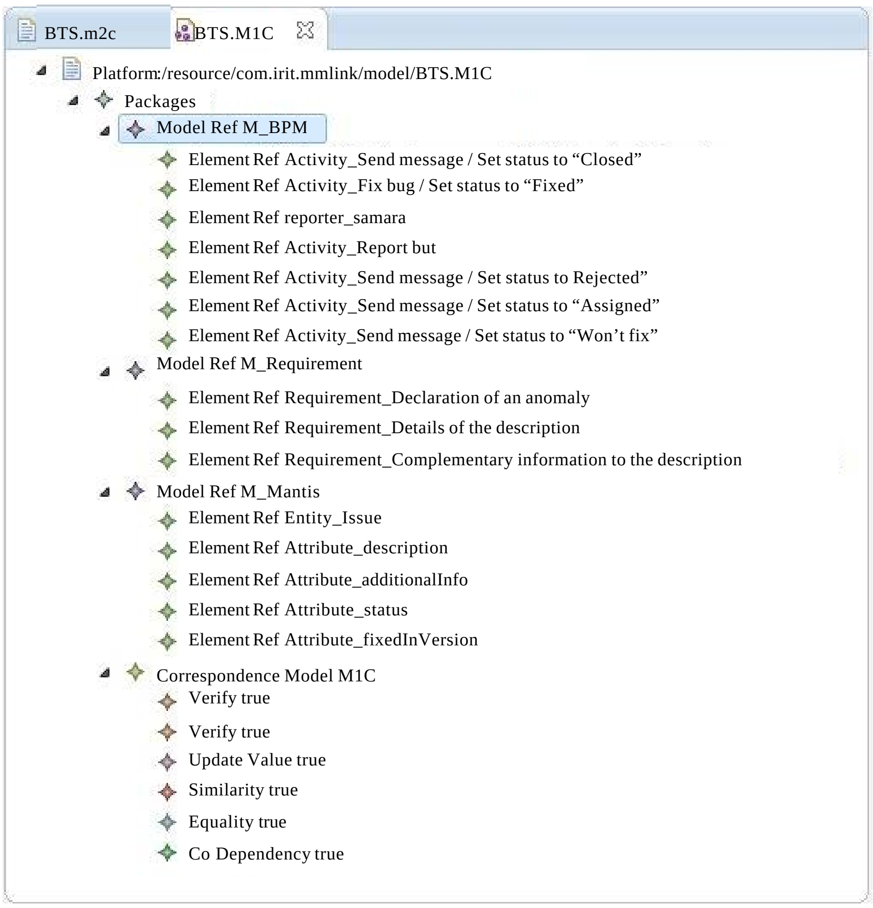
```
platform:/resource/POCModel_v2/model/BTS.m2c
  Packages
    Model Ref MM_BPM
      Element Ref Task
      Element Ref Lane
    Model Ref MM_Requirement
      Element Ref requirement
    Model Ref MM_Mantis
      Element Ref Attribute
      Element Ref Enity
    Correspondence Model M2C
      Verify true
        Expression OCL
      Update Value true
      Similarity true
      Similarity false
      Dependency true
```

Properties	Value
Body	invariant
Language	OCL

Fig. 18: Example of M2C for the BTS domain

Once the M2C model is created, the M1C can be obtained by choosing the appropriate refine type as explained in section 0.

To implement the refine relation, we need transformation rules created using languages such as ATL (Wagelaar, 2005) or QVT (OMG, 2011b). The transition from HLC to LLC is horizontal as shown in Fig. 11. However, this transition is vertical as well. In fact, HLC relate meta-elements whereas LLC relate model elements. Refine is therefore a hybrid transformation. Consequently this



BTS.m2c
BTS.M1C

```
Platform:/resource/com.irit.mmlink/model/BTS.M1C
  Packages
    Model Ref M_BPM
      Element Ref Activity_Send message / Set status to "Closed"
      Element Ref Activity_Fix bug / Set status to "Fixed"
      Element Ref reporter_samara
      Element Ref Activity_Report but
      Element Ref Activity_Send message / Set status to Rejected"
      Element Ref Activity_Send message / Set status to "Assigned"
      Element Ref Activity_Send message / Set status to "Won't fix"
    Model Ref M_Requirement
      Element Ref Requirement_Declaration of an anomaly
      Element Ref Requirement_Details of the description
      Element Ref Requirement_Complementary information to the description
    Model Ref M_Mantis
      Element Ref Entity_Issue
      Element Ref Attribute_description
      Element Ref Attribute_additionalInfo
      Element Ref Attribute_status
      Element Ref Attribute_fixedInVersion
    Correspondence Model M1C
      Verify true
      Verify true
      Update Value true
      Similarity true
      Equality true
      Co Dependency true
```

Fig. 19: Example of M1C for the BTS

refine transformation cannot be implemented using classical transformation languages as they don't address the hybrid aspect. To solve this problem we have written the refine transformation in Java/EMF.

Figure 19 shows an example of M1C produced by a refine of type extension. This model includes the correspondences previously defined in Figure 8. The two new relationships "CoDependency" and "Equality" are extensions, respectively "Dependency" and "Similarity" ones.

RELATED WORKS

Several research works are related to models matching. In AMW (Del Fabro *et al.*, 2005) authors describe a language that allows using M2M transformations for model comparison. However, AMW is usable only when source and target model are very similar and developers must add extensions to the metamodel, so as to permit the definition of relationships, even for the obvious ones (like similarity). Moreover according to Kolovos (2009) transformations been used are generally verbose as they need to compensate for the fact that M2M language do not provide tailored constructs for the task of model comparison. To optimize the representation of a composed model, authors of the same team propose a model virtualization technique (Clasen *et al.*, 2011a). Such technique may be useful for implementing our approach, especially models tracing and impacts calculation in case of source models evolution.

ECL (Kolovos *et al*., 2006b) is a matching language which is difficult to use because it requires specialized skills and great efforts, since correspondences are produced by executing rule based algorithms written manually. It also compares only one source element with one target one. Moreover, the result of the matching operation is a trace of correspondence which contains the needed relationships after performing a set of rules. To exploit that trace and be able to reuse the result for MDE purposes (e.g., composition), the developer must perform a serialization step to transform the traces into a model of correspondences.

The Kompose approach (Drey *et al*., 2009) addresses the composition of homogeneous source models. The process of matching must be parameterized by defining signatures at the metamodel level in order to define specific matching operators. In this approach, heterogeneity of models is not taken into account yet.

MatchBox (Voigt *et al*., 2010) transforms input models into a tree model called AMC (Auto Mapping Core). The process continues by applying a set of matching strategies to produce the model of correspondences. The major drawback of MatchBox is that it is the developer task to define transformations to the AMC model. Another disadvantage raised by the authors, is the loss of information during the transformations.

EMF Compare (Brun and Pierantonio, 2008) calculates the correspondences on the basis of the similarity. The matching engine is based on heuristics and elements are compared using several metrics including: similarity of name, content type and relationship. The values returned vary from 0 to 1 which will be pooled together to obtain overall similarity values. In the same context, DSMDiff (Lin *et al*., 2007) extension of the work described in Xing and Stroulia (2005) which uses name and structural similarities to identify the correspondences between UML models, has adopted the same calculation technique with the advantage of supporting different DSLs specified via GME (Generic Modelling Environment). Both approaches use only similarity and they do not exploit other relationships.

In general, studied matching approaches have shortcomings at two moments of the matching process: before and after the creation of the correspondence model. Regarding the first moment, we can notice the lack of balance between the ability to express correspondences and their reusability. Existing approaches are based mainly on only one of these criteria, as reusability comes at the price of less expressiveness and vice versa. In addition, these approaches manage only binary correspondences and therefore cannot establish complex n-ary ones relating a model element to any set of elements belonging to other models. Concerning the second moment, we can note that studied approaches produce a correspondence model between each pair of input models; so for n input models, [n×(n-1)]/2 correspondence models must be created which leads to a large number of separate models without any connection between them and which makes their management very difficult and almost impossible to automate. In our approach, we produce a unique model of correspondences among input models.

CONCLUSION AND PERSPECTIVES

Our general research work addresses the matching of interrelated heterogeneous models in the context of complex systems development. Thereby, we are interested in establishing correspondences between heterogeneous models described through different DSLs corresponding to different business areas of a domain. In this study, we propose a process to establish correspondences between such heterogeneous source models via., a matching operation (semi-automatic) based on a correspondence metamodel (MMC). The generic part of MMC captures correspondences based on independent domain semantic relationships. MMC can be thus extended through specialization of the "domain specific relationship" meta-class according to specific domain. Relationships among source models are identified first at the metamodel level and then refined at the model level.

Compared to the presented approaches that deal with the same problem, our approach proposes several distinguished qualities:

- **Commonality:** MMC provides a "generic" part-common to all domains-that defines a description of most common relationships. For so, we run a survey on different DSLs belonging to different domains in order to identify the frequent used ones, to avoid their specification repeatedly on the MMC
- **Variability:** MMC can be extended depending on the pecularities of the domain under consideration, in order to support the relationships relating to specific business areas where the domain is complex. This is done through specializations of the "domain specific relationship" meta-class
- **Flexibility:** Thanks to flexibility at the conceptual level, the MMC can relate n models (through their model elements) and express n-ary cardinality for each possible correspondence
- **Light weight:** The MC (model of correspondence conform to the MMC) is built in a virtual manner (Clasen *et al.*, 2011a, b). Indeed the MC contains only the important concepts. Elements are accessible through references and they don't have a physical existence in the MC.

There are several lines of work in which we are currently engaged. Firstly, as we explained previously, all the correspondences are created by the domain expert through the HMS tool. In order to alleviate the task of the expert by automating the creation of certain correspondences, predominately the Domain Independent Relationship (DIR), we will use a linguistic resource: Wordnet. Wordnet is a lexical database with hundreds of thousands of meanings which are connected by complex architecture relation such as, hyponymy, meronymy, entailement. Using Wordnet will allow us to find different correspondences whose relationships are of DIR kind as some of it connections are similar to them.

Secondly, we intend to complete the development of HMS by adding the graphical syntax and the semi-automatic definition of correspondences. Lastly, our approach of matching has a wider application than just defining correspondences between different models. In fact the objective of the proposed approach is not simply to relate different models but to compose them virtually and to keep them synchronized. For this we intend to exploit the correspondence model to address some maintenance issue in case where source models evolve. Our goal is then to provide a semi-automatic collaborative process allowing to (1) Update the M1C model, (2) Calculate impacts of a change occurred in a given source model and (3) Propose modifications to maintain the consistency of the system.

ACKNOWLEDGMENTS

This study describes the results of a research work in the scope of the PHC Volubilis MA/11/258. We would like to thank Povilas Safranauskas, account executive into No Magic, Inc. for his helps on Magicdraw, as it is the tool used to represent the different models of this work.

REFERENCES

Anwar, A., S. Ebersold, B. Coulette, M. Nassar and A. Kriouile, 2010. A rule-driven approach for composing viewpoint-oriented models. J. Object Technol., 9: 89-114.

Boulanger, F., C. Jacquet, C. Hardebolle and E. Rouis, 2010. Modeling Heterogeneous Points of View with Modhel'x. In: Models in Software Engineering, Ghosh, S. (Ed.). Springer, Berlin, Germany, pp: 310-324.

Brun, C. and A. Pierantonio, 2008. Model differences in the eclipse modelling framework. Eur. J. Inform. Prof., 9: 29-34.

Cariou, E., N. Belloir and F. Barbier, 2009. Contrats de transformation pour la validation de raffinement de modeles. 5e^{mes} Journees sur l'Ingenierie Dirigee par les Modeles (IDM 2009), March 2009, http://web.univ-pau.fr/~ecariou/papers/idm09.pdf

Castano, S. and V. De Antonellis, 2001. Global viewing of heterogeneous data sources. IEEE Trans. Knowl. Data Eng., 13: 277-297.

Clasen, C., F. Jouault and J. Cabot, 2011a. Virtual composition of EMF models. 7emes Journees sur l'Ingenierie Dirigee par les Modeles (IDM 2011), Lille, France, http://hal.archives-ouvertes.fr/inria-00606374/.

Clasen, C., F. Jouault and J. Cabot, 2011b. VirtualEMF: A Model Virtualization Tool. In: Advances in Conceptual Modeling. Recent Developments and New Directions, De Troyer, O., C.B. Medeiros, R. Billen, P. Hallot, A. Simitsis and H. Van Mingroot (Eds.). Springer, Germany, pp: 332-335.

Del Fabro, M.D., J. Bezivin, F. Jouault, E. Breton and G. Gueltas, 2005. AMW: A generic model weaver. IDM-Ingenierie des Modeles, 2005, Paris, 1eres Journees sur l'Ingenierie Dirigee par les Modeles, 2005.

Drey, Z., C. Faucher, F. Fleurey, V. Mahe and D. Vojtisek, 2009. Kermeta language. Reference Manual, April 10, 2009, http://www.kermeta.org/docs/KerMeta-Manual.pdf

Eker, J., J.W. Janneck, E.A. Lee, J. Liu and X. Liu *et al.*, 2003. Taming heterogeneity-the Ptolemy approach. Proc. IEEE, 91: 127-144.

El Hamlaoui, M., S. Ebersold, A. Anwar, M. Nassar and B. Coulette, 2013a. A process for maintaining heterogeneous models consistency through change synchronization. Proceedings of the ACS International Conference on Computer Systems and Applications, May 27-30, 2013, Ifrane, Morocco, pp: 1-4.

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

Epsilon, 2010. ModeLink. http://www.eclipse.org/epsilon/doc/modelink/

Eysholdt, M. and H. Behrens, 2010. Xtext: Implement your language faster than the quick and dirty way. Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion, October 17-21, 2010, New York, USA., pp: 307-309.

Fenza, G., V. Loia and S. Senatore, 2008. A hybrid approach to semantic web services matchmaking. Int. J. Approximate Reasoning, 48: 808-828.

Gronback, R.C., 2009. Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. 1st Edn., Addison-Wesley Professional, USA., ISBN: 978-0321534071, Pages: 736.

Hilliard, R., 2001. Viewpoint modeling. Proceedings of the 1st International Workshop on Describing Software Architecture with UML, May 13-15, 2001, Toronto, Canada.

Kolovos, D.S., R.F. Paige and F.A.C. Polack, 2006a. Merging Models with the Epsilon Merging Language (EML). In: Model Driven Engineering Languages and Systems, Nierstrasz, O., J. Whittle, D. Harel and G. Reggio (Eds.). Springer, Germany, pp: 215-229.

Kolovos, D.S., R.F. Paige and F.A.C. Polack, 2006b. Model comparison: A foundation for model composition and model transformation testing. Proceedings of the International Workshop on Global Integrated Model Management, May 20-28, 2006, Shanghai, China, pp: 13-20.

Kolovos, D.S., 2009. Establishing correspondences between models with the epsilon comparison language. Proceedings of the 5th European Conference on Model Driven Architecture-Foundations and Applications, June 23-26, 2009, Netherlands, pp: 146-157.

Koning, H. and H. van Vliet, 2006. A method for defining IEEE Std 1471 viewpoints. J. Syst. Software, 79: 120-131.

Lin, Y.H., J. Gray and F. Jouault, 2007. DSMDiff: A differentiation tool for domain-specific models. Eur. J. Inform. Syst., 16: 349-361.

MantisBT, 2010. Mantis bug tracker. http://www.mantisbt.org/index.php

Medvidovic, N. and R.N. Taylor, 2000. A classification and comparison framework for software architecture description languages. IEEE Trans. Software Eng., 26: 70-93.

OMG, 2003. MDA guide version 1.0.1. Document No. OMG/2003-06-01, Object Management Group, June 12, 2003.

OMG, 2008. OMG Systems Modeling Language (OMG SysML™), version 1.1. Object Management Group, November, 2008.

OMG, 2011a. Business Process Model and Notation (BPMN), version 2.0. OMG Document No. formal/2011-01-03, Object Management Group, January, 2011.

OMG, 2011b. Meta Object Facility (MOF) 2.0 query/view/transformation specification, version 1.1. OMG Document No. formal/2011-01-01, Object Management Group, January, 2011.

Ober, I., B. Coulette and Y. Lakhrissi, 2008. Behavioral modelling and composition of object slices using event observation. Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems, September 28-October 3, 2008, Toulouse, France, pp: 219-233.

Steinberg, D., F. Budinsky, M. Paternostro and E. Merks, 2009. EMF: Eclipse Modeling Framework. 2nd Edn., Addison-Wesley, USA.

UML, 2007. OMG Unified Modeling Language (OMG UML), superstructure, V2.1.2. Unified Modeling Language, November, 2007.

Voigt, K., P. Ivanov and A. Rummler, 2010. Matchbox: Combined meta-model matching for semi-automatic mapping generation. Proceedings of the ACM Symposium on Applied Computing, March 22-26, 2010, Sierre, Switzerland, pp: 2281-2288.

Wagelaar, D., 2005. Context-driven model refinement. Proceedings of the European Conference on Model Driven Architecture: Foundations and Applications, June 26-27, 2003, Netherlands, pp: 189-203.

Wolf, W.H., 1994. Hardware-software co-design of embedded systems [and prolog]. Proc. IEEE, 82: 967-989.

Wooldridge, M., 1997. Agent-based software engineering. Proc. Software Eng., 144: 26-37.

Xing, Z. and E. Stroulia, 2005. UMLDiff: An algorithm for object-oriented design differencing. Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, November 07-11, 2005, Long Beach, CA., USA., pp: 54-65.

Zito, A., Z. Diskin and J. Dingel, 2006. Package merge in UML 2: Practice vs. theory? Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems, October 1-6, 2006, Genova, Italy, pp: 185-199.