An Approach to Models' Comparison based on their Semantics

Jamal Abd-Ali, Karim El Guemhioui

Department of Computer Science and Engineering
University of Quebec in Outaouais (UQO)
C.P. 1250 succ. Hull, Gatineau QC, J8X 3X7 Canada
{abdj01, karim}@uqo.ca

Abstact. Whenever a model is subjected to a model transformation, we are faced with the problem of validating the compliance of the output model with the input one. This paper tackles the issue of comparing models bound to the same development process or exhibiting a potential of resemblance that can be exploited. The models' comparison is based on the semantics of the metamodel elements used to express the models. These semantics must be completely and exclusively expressed in single elements. Otherwise, these elements must be rewritten so as to extract specific semantics that are embedded within larger ones, or to consolidate particular semantics that are diluted in more than one element. We propose a step-by-step approach to conducting models' comparison based on their semantics; and we provide metrics to quantify the outcome of the comparison and to support the parameterization and automation of the comparison process.

1 Introduction

In a model driven development process, the unfolding of the work is characterized by a synergy between several models, generally written in different languages. Each time that a model is transformed, or even that a new one is created, the confrontation of the new model with the previous ones can challenge the coherence of the various development process stages. For example, in an MDA approach where a Platform Independent Model (PIM) is to be transformed into one or more Platform Specific Models (PSM), or a PSM into another PSM, we are faced with assessing the compliance of the output model to the input one. Moreover, the comparison of an application specification model to its implementation model is still an open research topic.

Besides, the re-use of a model M or of a part of M, requires a rewriting of M in a new language allowing an integration of M into other models written in this latter language. A rather classical example is that of the transition of a relational representation to an object-oriented one; it implies a transformation of models, known under the name of Class-Relational, which does not offer any means of comparison of its output models with its input models.

In the remainder of this paper, we tackle the issue of model comparison by focusing on the semantics attached to the meta elements used in the writing of the models. We propose a step by step approach that enables model comparison of models initially expressed in different metalanguages. We define a set of metrics to assist the developer in quantifying the outcome of the comparison. Finally, we discuss the contribution of the proposed approach and conclude with plans for future work.

2 Related Work

Our work on semantics based model comparison is inspired by a seminal work by Kleppe and Warmer on preserving semantics. The authors explain that given a source model, this model describes a system. In other words, the meaning of a model is the system it describes. A model transformation from a source model to a

target model is defined such that the target model must describe the same system. Therefore, the target model must have the same meaning [6]. Most of the work on model comparison found in the literature falls into one of three approaches. We briefly summarize each one of them through some recent, tough representative, articles.

(1) The first approach uses a framework of formal languages, like the Constructive Theory Type [11], or the algebraic specification language CASL [2, 3]. It relies on these languages to express models and metamodels before submitting them to a formal validation. However, this kind approach is short in providing a model comparison method that goes beyond the structure of the metamodels to support their underlying semantics.

(2) A second approach deals with executable models and is based on probabilistic computations that use data collected from repetitive executions of the two models, and confronts their behaviors with the data provided by these experiments [10]. An illustration of this approach would be to compare the model of a set of non functional constraints, which is not executable, to its implementation model, which is generally an executable and testable model. The approach will offer a means to verify how much one model complies with the other based on execution feedback. Note however that, in theory, it is impossible to use testing as a method to establish whether two systems are the same, because we can never test all possible inputs against all possible outputs [6].

(3) A third approach is based on intuitive steps related to the domains of the models under consideration, and that take into account the developer expectations regarding the comparison of these models. The approach uses a combination of several complimentary validation techniques [12] which comprise: Establishing a mapping based on a deep understanding of both metamodels; Specifying a set of structural rules that govern comparable models; Testing behavioral compliance using appropriate tools that can generate sequence diagrams in order to compare them. This third approach, though far from offering formal evidence of model compliance, relies on some sound principles. It compares models based on a set of correspondences between fragments of models. The fragments are specified using various modeling layers, and they reflect the semantic correspondences of interest to the developer. Our approach complements this last category.

3 The Metamodeling Foundations of our Approach

When transforming models at the same level of abstraction (e.g., PSM to PSM) or across abstraction levels (e.g., PIM to PSM), we are faced with the issue of preserving the compliance of the output model with the input one; that is, assessing the quality of the transformation.

3.1 Basics of Model Comparison

Consider two models M1 and M2 written with an expression language modeled, respectively, by the metamodels MM1 and MM2. For commodity, we will abusively say that M1 and M2 are, respectively, written with MM1 and MM2.

We are interested in comparing M1 and M2 inasmuch as the systems Syst1 and Syst2, represented by M1 and M2, have some semantics in common. Comparing M1 and M2 boils down to comparing the semantics they convey and, ultimately, the systems Syst1 and Syst2 they represent.

The semantics attached to metamodel elements are the bridge that links the elements of a model to what these elements represent in a real system. For example, a single system will undoubtedly be described differently if we resort to two distinct metamodels, because the system's semantics will be allocated to different elements in each metamodel. This means that whenever two models are written with two different metamodels, the closeness or the divergence between the two modeled systems won't usually be manifest at the model level. Therefore, we need to carefully consider the language of expression (i.e.; the metamodels) of the two models we are interested in comparing.

In our approach to model comparison, we consider resorting to a single expression language (i.e.; metamodel) which will enable the identification of common semantics noticeable in M1 and M2. These semantics

are of course those of the metamodels' elements belonging to MM1 and MM2. In the remainder of this article, these semantics will be called *shared semantics* and they will guide our approach.

Note that one can always consider a metamodel whose elements do not have determined semantics. Such metamodel and the models it describes are said to be *abstract*. This makes it possible for this metamodel to be interpreted according to the semantics one chooses to assign to its elements. An example of an abstract model would be a graph of nodes and edges without any semantics that relates them to a specific system. Such model would earn some semantics if it was specified that a node represents a state and an edge represents a transition. Another realization of semantics would be for the nodes and edges to represent, respectively, cities and connecting roads.

3.2 Hypotheses and Notations

In this paper, we are mainly interested in cases where metamodels' elements are endowed with semantics that can guide and explain the process of comparing models described by these metamodels.

We also limit ourselves to MOF based models so as to clearly and convincingly convey the feasibility of our approach. This does not hamper the generic aspect of the approach because this latter is not intrinsically dependent on MOF and/or MOF related languages.

The metamodels and models considered are finite; the number of elements of each one is finite. Below, we remind and complement the notations and conventions we have adopted:

- M1 and M2 are the two models considered for a comparison; their metamodels are respectively noted MM1 and MM2.
- Syst1 and Syst2 are the two systems represented, respectively, by M1 and M2 (in the MDA meaning of a system represented by a model).
- We convene to (abusively) call instance of a metamodel any model written with this metamodel.
- When dealing with elements, and sets, and their relationships, we will comply with the notations of the set theory.
- We convene to (abusively) call instance of a metamodel any model written with this metamodel.
- We mean by element of a metamodel any instance of Class, Association, or Data Type of MOF [7]; that is any entity that represents a type (i.e.; MOF Classifier).
- All of the elements of a metamodel form a set. An individual element is noted T_k, where k is an index identifying the element in the set.
- All of the instances of the various T_k elements, contributing to the expression of a model, form also a set.
- A fragment of a model is any subset of this model.
- When no ambiguity can result, we will confuse the models (and/or metamodels) with their sets of elements, to express relations of inclusion, equality, union, subtraction, disjunction, and membership, as defined in the set theory.
- We call S_i (or S_j) any semantics identifiable at a metamodel level; with i (or j) = {1, …, n}.
- We call developer, the person who expresses her interest in the comparison of models according to their semantics and who carries out the process.

3.3 Objectives and Milestones

Based on the rationale underlying our approach to model comparison, we aim at achieving the following main goals:

1. Flesh out a metamodel MM0 that supports the rewriting of the two models M1 and M2. MM0 will supply the elements representing the shared semantics of both metamodels MM1 and MM2. MM0 will provide the expression language able to replace those of MM1 and MM2.
2. Define a set of metrics to assist the developer in assessing the closeness of or the divergence between the two models M1 and M2.

In accomplishing the aforementioned goals, the following tasks will be carried out:

1. Detect all the model fragments that embody the same semantics in the two models being compared, namely, M1 and M2. We will say that there is a semantics correspondence.

2. Use the aforementioned metrics to guide the process of detection of correspondences according to the developer's assumptions and purposes, so as to filter out the results which could otherwise be rather bulky.
3. Detect any incoherence between the two models M1 and M2.
4. Test if one of the models can be deduced from the information contained in the other.
5. Test if the two models share some property that can be expressed at the level of the MM0 elements.

3.4 Elaboration of the MM0 Metamodel

The process of comparing two models starts at the level of their metamodels. The first step is to create a third metamodel MM0 which can replace MM1 and MM2 for expressing the two models M1 and M2 we want to compare. As explained in section 3.1, this substitution will uncover elements' semantics at the model level, and therefore enable the detection of semantics correspondences between the two models. Note that the MM0 metamodel will be reusable for any comparison of models based respectively on MM1 and MM2.

Initially, MM0 consists of MM1 and MM2, considered as two distinct packages (in the MOF sense). Thus, M1 and M2 will be two instances of the same metamodel MM0. Of course, the problem has now become more complicated because MM0 lacks cohesion and is much heavier to handle due a larger number of elements. A refinement of MM0 is necessary to render it exploitable for a model comparison. This refinement process is iterative and consists in:

- Drawing up a list of all the semantics S_i shared by the elements of both metamodels MM1 and MM2. They represent the concepts that interest the developer in her comparison of M1 and M2.
- Classifying the semantics of MM1 and MM2 elements as follows:
  - Group 1: comprises any semantics S_i such as there are two single (unique) elements $E1_i$ of MM1 and $E2_i$ of MM2, each one representing only the S_i semantics to the exclusion of any other semantics.
  - Group 2: includes any shared semantics S_j which does not belong to Group 1. To be more explicit, several scenarios can contribute to this situation: (1) At least one of the two elements representing the S_j semantics, respectively, in MM1and MM2, embodies also some additional semantics. We will say that S_j is represented (either in MM1 or MM2 or in both) within a larger semantics - (2) At least in one of the metamodels MM1 or MM2, the S_j semantics is not represented by a single element but by the association of several elements. We will say that each element in the association represents a part of the S_j semantics. (3) A combination of (1) and (2).
  - Group 3: gathers semantics not shared by both metamodels MM1 and MM2. This last group won't require further processing because it represents concepts not found in common in both metamodels. Any attempt at comparing models with respect to semantics in this last group is vain.
- Processing of Group 1: For each S_i in this group, there are two elements $E1_i$ and $E2_i$ which embody this semantics. Therefore, in the MM0 metamodel, we will eliminate $E2_i$ (for example) and keep only $E1_i$ to represent the S_{1i} semantics. This coalescence of pairs of elements with shared semantics will ensure that elements common to MM1 and MM2 are integrated into MM0.
- Processing of Group 2: For each S_j in this group, if in neither MM1 nor MM2 there is no single element that represents this semantics and solely this semantics, we create a new E_j element in MM0 which will represent the S_j semantics distinctively from any other semantics. Otherwise, S_j is already distinctively and completely embodied in a single element of either MM1 or MM2 (but not both), and therefore of MM0. In addition, we consider each element of MM1 and/or MM2 representing S_j within a larger semantics (we call such element $E_{complex}$), and apply to it an operation we call a *split*. Note that in the case where the S_j semantics is represented by an association of elements (scenarios 2 and 3), $E_{complex}$ will be one of the elements in the association, usually the one we deem to represent the larger part of the S_j semantics. Anyway, the split operation aims at expressing the $E_{complex}$ relation to the S_j semantics, through a relationship with the (existing or newly created) E_j element. We will say that we have extracted the S_j semantics from $E_{complex}$ and made it manifest exclusively in the E_j element. In practical terms, the split operation entails a rewriting of the metamodel fragment containing $E_{complex}$ in such a way that $E_{complex}$ is replaced by a set of elements (newly created and/or already in MM0). This set either contains E_j or establishes a rela-

tionship with E_j, and none of the elements, with the exception of E_j, represents in any way the S_j semantics (they may however be dependent on this semantics via an association with E_j). Of course, all the associations, aggregations, references, or constraints that were involving $E_{complex}$ must be preserved by re-expressing them at the level of the elements in the replacement set. MM0 will no longer contain $E_{complex}$ (though the actual $E_{complex}$ name can be kept, but it no longer holds any part of the S_j semantics). The split operation aims to ensure that shared implicit semantics are explicitly, uniquely, and completely represented by a single element, avoiding thus any redundancy in MM0.

We repeat the split operation to all $E_{complex}$ elements until exhaustion of Group 2. Note that each split operation can, in turn, spawn several additional splits due to the fact that some of the newly created elements resulting from each split may themselves represent, in some way, a semantics we are interested in extracting. Thus, a split operation can depend on the results of previous splits targeting other elements. It may advantageous to defer spawned splits to some next iteration or even to the end (after having exhausted Group 2) for a better separation of problems. Anyway, the number of iterations is finite because the number T_k of metamodeling elements is finite. Based on our (short) experience, we recommend processing the MM0 Classifier instances in the following order: first Data Type, then Class, and finally Association instances.

Of course, we have to keep track of all the split operations, elimination of elements with duplicated semantics, creation of new elements, etc., in order to rewrite the M1 and M2 models in terms of the MM0 metamodel. The rewriting does not need to wait until the end of the processing of groups 1 and 2, but can proceed in parallel with the refinement of MM0.

3.5 Discussion of the MM0 Metamodel Elaboration

In our approach, we have purposely targeted elements that are instances of MOF Classifier because they represent the types of the elements of a metamodel. That is, they represent the essential vocabulary of the language defined by a metamodel, and they contain the other elements of a metamodel like instances of Attribute, Reference, Operation, etc.

Though we have explained how one proceeds to flesh out the MM0 metamodel, we haven't offered much detail regarding the rewriting of an $E_{complex}$ element (to extract and isolate a given semantics in a single element). This metamodeling activity, which depends on the shared semantics we are considering and its relation to the remaining semantic fields of both metamodels, is beyond the scope of this paper. However, one should not fear this task because simple manipulations involving, for example, inheritance, aggregation, and/or association will usually permit to push down, confine, and restrict a given semantics to a specific element.

Thus, we have limited ourselves to defining the split operation in terms of the expected results, rather than as a detailed implementation likely to be automated. However, once MM0 is obtained, the automation of the rewriting of the M1 and M2 models in terms of MM0 should be straightforward.

4 Metrics for Model Comparison

4.1 Preliminary Definitions

Two elements of M1 and M2 are said to have corresponding types, if either they are instances of the same element of MM0, or they are instances of two elements of MM0 which directly or indirectly specialize (through inheritance) the same element of MM0. In looking for correspondences, the developer can limit the search up to some level in the inheritance hierarchies of MM0 elements to avoid meaningless correspondences with respect to the semantics the developer is interested in comparing. Otherwise, we may end up concluding that we have a type correspondence whenever two elements share some remote abstract ancestor (such as Object or Component or other generic and usually semantics-less concept). Note that the correspondence of names between elements corresponding in type is not required, because these elements can

play in M1 the same roles as their corresponding types in M2, with a permutation of names or even with different names [13].

A fragment of model, we call Frg_{M1}, is said to be *an M1-based correspondence fragment* between M1 and M2 if:

- Frg_{M1} is a fragment of model of M1;
- Each element e1 of Frg_{M1} has a corresponding type element e2 in M2; we say that e2 is the image of e1 with respect to Frg_{M1};
- For each association or aggregation connecting two elements e1 and e'1 of Frg_{M1}, there is another corresponding in type construct connecting the images e2 and e'2 in M2;

The image of Frg_{M1} is an M2 model fragment made of all the images of the Frg_{M1} elements as well as all the associations connecting them in M2. This image is clearly an M2-based correspondence fragment between M1 and M2 (i.e.; Frg_{M2}).

We call a correspondence fragment between M1 and M2 (without specifying if it is based in M1 or M2), any fragment of model obtained by a conciliation of Frg_{M1}. This conciliation consists in doing the following. For each element e1 of Frg_{M1} whose image in M2 is e2, we set to null any attribute, reference, constraint, and/or contained element, whose value differs from the one in its image e2. For example, if an attribute is having different values in e1 and e2, this attribute is set to null in the element e1 of the correspondence fragment between M1 and M2. Note that this conciliation is only needed for the computation of our metrics; thus it need not be destructive of M1: a copy of the correspondence fragment can be made for conciliation purposes.

Definition 1: We call weight of a metamodel element T_k, a positive real value m_k assigned to this element by the developer to represent the importance she wishes to attach to the semantics of this element. The sum of all the m_k for all the T_k of the metamodel is equal to 1.

Definition 2: The weight of a T_k instance is equal to m_k.

Definition 3: we call expressiveness of a model M, the sum of the weights of all its elements. It is noted Expr(M).

4.2 Metrics

To quantitatively evaluate the comparison between M1 and M2, we will rely on M_{int} which is an intersection model between M1 and M2 that has as many as possible corresponding types.

M_{int} is formed by the correspondence fragment between M1 and M2 whose expressiveness is maximum. Note that M_{int} may actually contain several disjoint correspondence fragments, provided M_{int} constitutes, in turn, a correspondence fragment between M1 and M2 (such as defined in the previous section). Moreover, the developer can always impose additional constraints on the correspondence fragments; for example: she may require that M_{int} contains specific elements representing chosen couples of elements corresponding in type in M1 and M2.

The elements imposed by the developer reflect her perceived correspondence in meaning between the two models; they constitute a starting core of a correspondence fragment between M1 and M2, leading to M_{int} with maximum expressiveness.

Definition 4: The distance between two models M1 and M2, whose intersection model is M_{int}, is equal to the smaller of Expr(M1) and Expr(M2), decreased by Expr(Mint). This metric provides a means to quantify the importance of the elements not belonging to M_{int} and preventing a total correspondence between M1 and M2. It is noted D(M1, M2), with $D(M1, M2) = \min(Expr(M1), Expr(M2)) - Expr(M_{int})$.

Definition 5: The divergence between two models M1 and M2 of non null expressiveness is equal to the distance between M1 and M2 divided by the expressiveness of the model with the smallest expressiveness. It is noted Div(M1, M2), with $Div(M1, M2) = D(M1, M2) / \min(Expr(M1), Expr(M2))$.

4.3 Characteristics of the Metrics

The defined metrics have some properties that justify their existence and bring to light the advantages they offer to the developer in her endeavor to compare two models. Below, we mention the most relevant for our work:

- Two identical models will have a null divergence between them: Div(M1, M1) = 0. The reverse is not always true, unless the developer forbids null weights for MM0 elements and for any of their contained elements.
- Two models which do not have any corresponding types, when considering types with strictly positive weights, will have a divergence equal to 1. In other terms: ((M_{int} contains only elements whose weights are null) or (M_{int} is empty)) $\Leftrightarrow$ Div(M1, M2) = 1 .
- By definition: Div(M1, M2) $\in$ [0,1]. This allows an immediate interpretation of the computed result, whatever the weighting chosen by the developer.
- Partial derivative of Div(M1, M2) with respect to the number of couple of elements with corresponding types, is equal to the weight of the considered type divided by the number of instances of the type; the whole divided by the expressiveness of the model with the smallest expressiveness. This gives a quite precise significance to the weight the developer assigns to an element in order to evaluate its importance in the model comparison. According to the previous property, the developer can determine the partial derivative of the divergence metric by affecting to MM0 elements, real positive weights between 0 and 1.
- Since semantics are at the heart of the comparison between M1 and M2, the developer must have a good grip of the divergence computation strategy, so as to give more importance to some semantics compared to others. As explained in the previous section, the developer has some slack in the construction of M_{int} since she can impose some constraints based on her knowledge of the rules that govern a valid semantic correspondence between M1 and M2.
- Note that an appropriate weighting makes it possible to directly deduce from D(M1, M2), the number of elements with corresponding type in M1 and M2 which are instances of a T_k element chosen by the developer.
- Once the weighting is done by the developer, the calculation of the divergence can be automated using algorithms of complexity not exceeding the resource capacities of nowadays machines.
- A relation of order can be defined on a set of models by considering the distances from each one to a specific model M_R taken as a reference. This makes it possible to classify the models according to intervals of distances separating them from M_R.

Furthermore, the computation of the divergence can also be made dependant on the correspondence of the elements that may be contained in the corresponding type elements of M1 and M2. Indeed, the expressiveness of M_{int} can take into account the type correspondences at a finer granularity. We can view the weight of a T_k metamodel element as the sum of the weights of the elements contained in T_k. Thus, the weight of T_k = $(\sum_p C_{kp}) * m_k$ where C_{kp} is the weight contribution of each element p contained in T_k such as $\sum_p C_{kp} = 1$.

At the model level, the weight of an instance t_{ki} of T_k is equal to m_k multiplied by the sum of the C_{kp} contributions of the elements contained in t_{ki}, with each one of these contributions multiplied by an adjustment factor $delta_{kp}$; $delta_{kp} = 0$ if the element concerned is to be put to null, $delta_{kp} = 1$ otherwise. Thus, the weight of an element $t_{ki} = (\sum_p delta_{kp} * C_{kp}) * m_k$. This way, if an element e of a correspondence fragment, which corresponds in type to its image but without a correspondence of one of its contained elements, this contained element is affected a null value. This involves a $delta_{kp} = 0$ which reduces the weight of the element e and consequently decreases the expressiveness of the correspondence fragment. This, in turn, will result in a search for a correspondence fragment of larger expressiveness to form M_{int} or will result in a smaller evaluation of Div(M1, M2).

Finally, to illustrate the control the developer has on implementing the computation of divergence between M1 and M2, we can mention, among the constraints or assumptions that she can impose on the process, the fact that she can require that any correspondence fragment be conform to MM0 in addition to some constraint, such as for example to withdraw an instance of a certain type if it is not associated in the correspondence fragment to another instance of some other specific type chosen by the developer. This fits well the requirements for comparison of models based on their semantics, because there may be elements that are

meaningless if they are not part of some structure of elements; in other term, if they do not conform to a pattern.

4.5 Other Uses of MM0 and the Proposed Metrics

At this stage, we dispose of the two models M1 and M2 expressed in MM0 terms and exhibiting all the shared semantics in the form of atomic elements of MM0. To benefit further from MM0 and the defined metrics, we can use them to explore the detection of inconsistencies between M1 and M2, and to check gain and/or loss of information resulting from a substitution of M1 for M2 or conversely:

- Fusion of M1 and M2: one can look at M1 and M2 as only one model, called M_{fusion}, and having in common the intersection model M_{int}. This new model can be examined by any tool which incorporates the metamodel MM0. The purpose is to examine whether this model presents inconsistencies or non conformities with the MM0 requirements.
- Use of OWL: M1, M2, and M_{fusion} are written according to the MM0 metamodel. They can all be written in OWL, according to a recent work related to MOF [9]. We will then have tools (reasoners) supporting OWL which will allow us to check for inconsistencies and other properties, and to use any kind of inference to query M1 and M2. These tools, such as Vampire, can even lead to conclude that M2 makes it possible to deduce M1 (or the reverse) to deal with a loss of information problem when going from M1 to M2 (in an MDA model transformation).

5 Illustrating Examples

Example 1: Classes and Tables

Consider two simplified metamodels [4], MM1 for object orientation, and MM2 for relational databases. Figure 1 shows them before any manipulation.

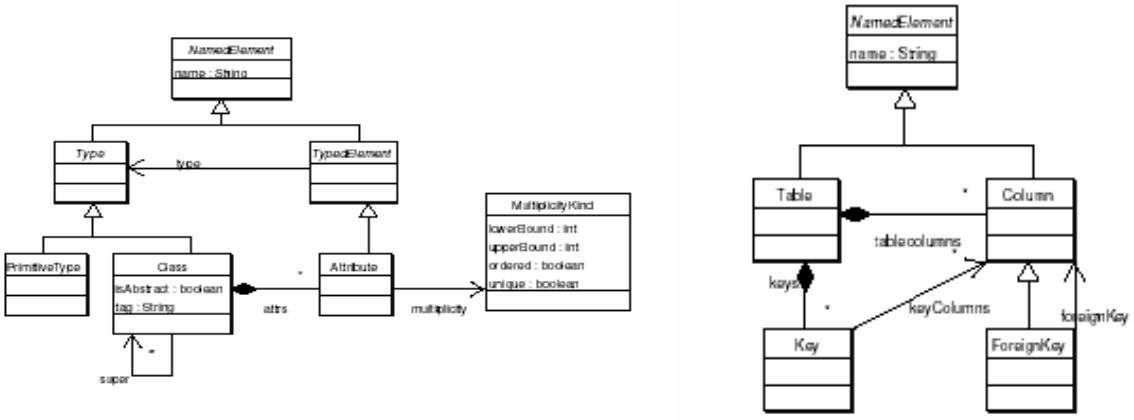

Fig. 1. A simplified representation of the UML and RDBMS metamodels [Trans]

A semantics S of interest to us in a comparison of instances of these two metamodels, is the concept of *structure* which encapsulates fields in only one type. The Class element of MM1 contains this semantics within a larger semantics specific to the classes. Thus S is of Group 2 according to our classification, and the Class element will undergo a split operation which will extract S from it to separately represent it by a single element that we will call Structure.

Similarly, in MM2, the Table element will be associated to the new Structure element rather than implicitly containing the S semantics. A second semantics S' for the concept of *structural feature* must supplement S. S' is partly embodied in the Attribute and Column elements of MM1 and MM2, respectively. Thus, S' belongs to Group 2. In elaborating MM0, Attribute and Column will each belong to a fragment of model where the representation of S' has been confined to a new element we will call StructuralFeature. Attribute and Column will explicitly represent the S' semantic by specializing the StructuralFeature element (inheritance).

This example clearly shows that the split operation does not follow a predetermined algorithm, but involves a metamodeling process that rewrites the elements Class and Attribute of MM1, and Table and Column of MM2, in a way that guaranties that the new elements Structure and StructuralFeature are the exclusive representatives of the S and S' semantics. The addition of the two elements Structure and StructuralFeature in the MM0 metamodel will replace the part of the elements Table and Class related to those semantics. Figure 2 shows the MM0 metamodel after the aforementioned splits.

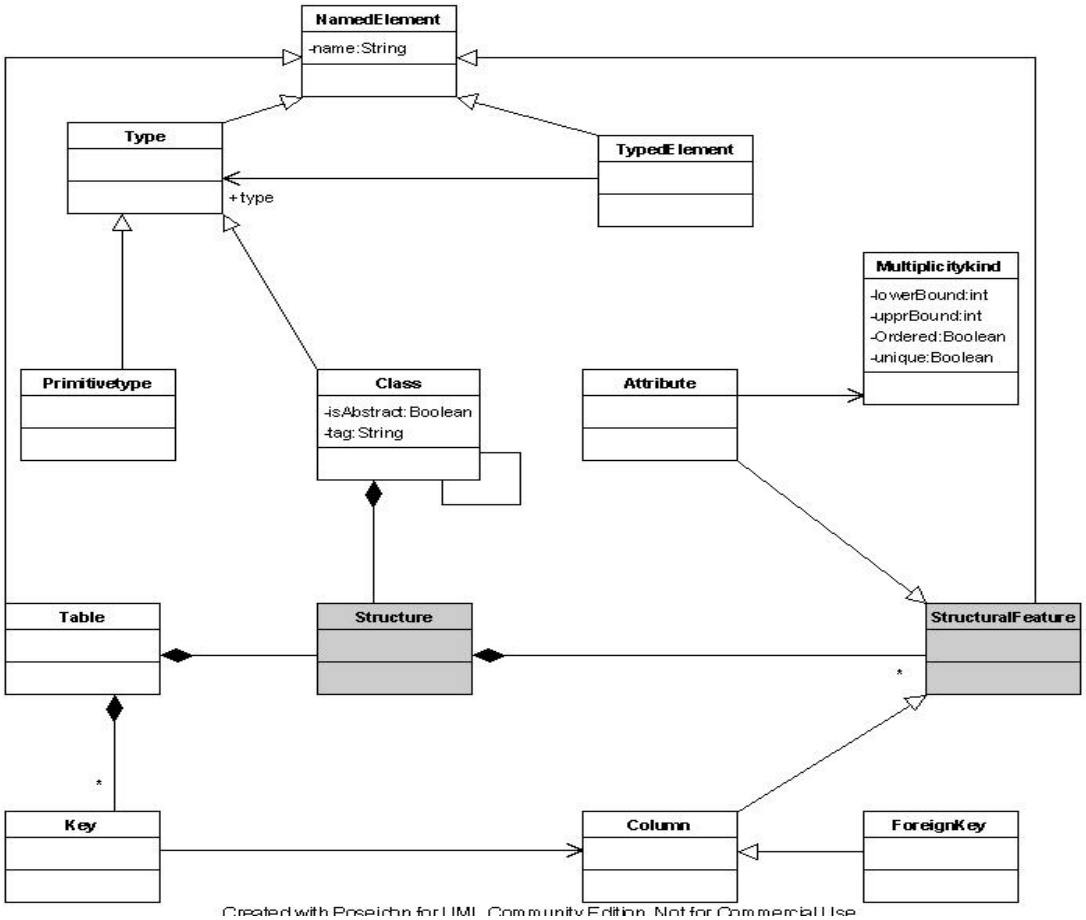

Fig. 2. MM0 enriched with 2 new elements for OO and RDBMS (shown in grey)

To illustrate the need to sometimes define additional developer's constraints, let us note that we must impose on MM0 that an instance based on StructuralFeature and contained in an instance of Structure which itself is an aggregate of an instance of Class, must necessarily be an instance of Attribute. Otherwise, it may inappropriately end up being an instance of Column.

Also exploiting MM0, if we are interested in seeking only the correspondence between Columns and Attributes belonging to corresponding classes and tables, we can use our metrics by applying a weighting which gives a strictly positive weight only to the StructuralFeature element (weight equal to 1). The developer can choose to ignore any type correspondence at the NamedElement type level (ancestor of Class and Table), because it just clutters the research of correspondence fragments.

Another way the developer can intervene in the comparison process, is by imposing constraints which put aside a type correspondence if the involved elements do not satisfy some condition of structure like conformity with a pattern (as mentioned in Section 4.3). As an illustration, we can require the coincidence of the StructuralFeature names and types, in addition to the correspondence of the Structure instances containing them.

Exemple 2: Two MDA PIMs

We consider two models, M1 and M2, each one describing components according to a different technology, namely JavaBeans and .Net Components. M1 is made of only one element, instance of EnterpriseBean and called EB1. The MM1 metamodel is that of the EJB components. Similarly, M2 is made of only one element, instance of NetComponent and called NC1. The MM2 metamodel is that of the .Net Components [1]. M1 and M2 are considered as two Platform Specific Models representing a same system within an MDA approach.

It is beyond of the scope of this paper to detail a complete elaboration of an MM0 metamodel which can describe both technologies EJB and .NET. For our demonstration, we suffice with a simplified situation in which each metamodel is limited to a single core element. The goal of the comparison is to explore the possibilities of validating a model transformation between the two technologies (EJB and .Net Components) by confronting the transformation entry model to its output model. The two models would totally incomparable unless we take into account the semantics of EnterpriseBean and NetComponent as elements representing the concept of component.

Thus, in the elaboration of the MM0 metamodel, we are faced with the detection of the concept of component common to both elements. Since the semantics commonly attached to the component concept is not uniquely and entirely attached to the elements EB1 and NC1, obviously this semantics belongs to Group 2. Thus, the need to create in MM0 a new element called Component, to represent uniquely and exclusively this semantics. EnterpiseBean will be replaced by the element Component in association with another element, called EJBParticularities, which represents the specific semantics of an EJB component (this semantics was implicitly contained in the EnterpriseBean element). In the same way, the NetComponent element will be split into Component and NetParticularities elements. MM0 will now hold the elements: Component, EJBParticularities, NETParticularities, as well as the remaining elements of the MM1 and MM2 metamodels. Figure 3 shows the MMo metamodel after the split operations.

The two models M1 and M2 are both expressed with MM0. They look as follows:

- M1 contains an instance of Component associated with an instance of EJBParticularities;
- M2 contains an instance of Component associated with an instance of NETParticularities.

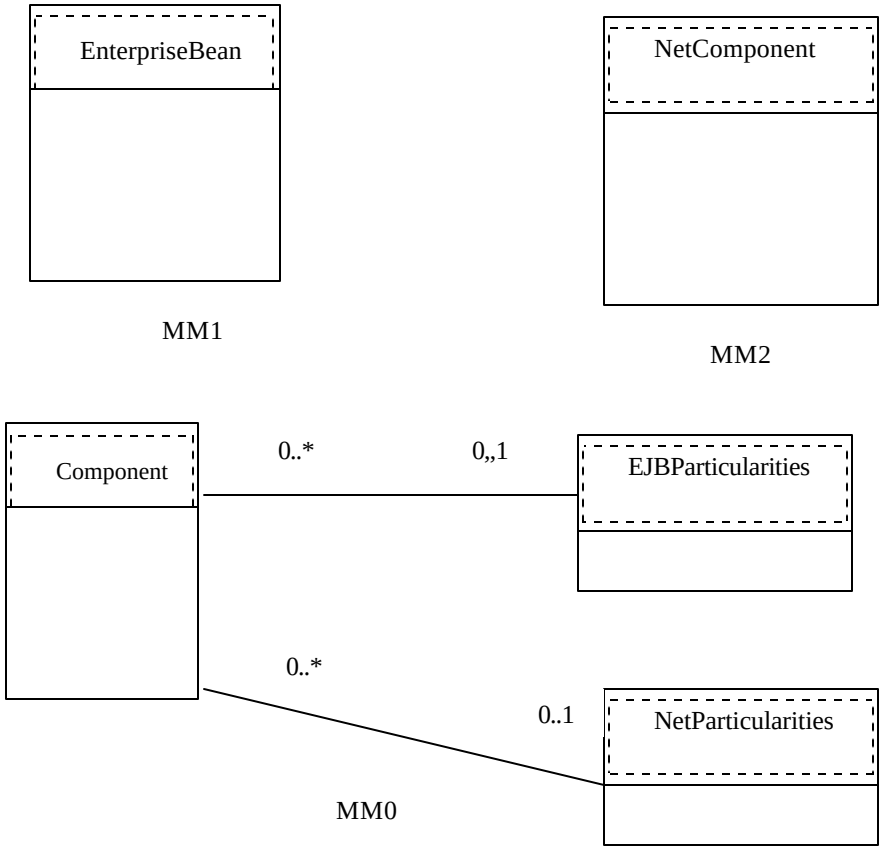

Fig. 3. Processing of the Component semantics

A second iteration would allow us to detect shared semantics now lying in the two elements EJBParticularities and NetParticularities. We could them proceed with a series of split operations to extract the semantics of pooling, session, deactivation, etc. Thus, the comparison of the two models can proceed yielding for finer correspondences at the component implementation level. Also, constraints can be added to MM0 to ensure that an instance of Component which is associated to an element specific to .Net, is not associated to an element specific to EJB.

6 Discussion of the Approach

6.1 Benefits of a Transitional MM0 Metamodel

The proposed approach to model comparison is in sync with other existing approaches (see Section 2) which seek to establish correspondences between patterns (model fragments) while relying on the semantics of both metamodels' elements. We believe that our approach goes beyond by supporting the extraction of sought semantics not represented in a model fragment. Our approach addresses the complex but very realistic situation in which a given semantics is spread across several complex concepts and represented in several elements of two metamodels MM1 and MM2, these latter being far from exhibiting direct correspondences. Our approach enables the separation of these concepts by assigning to them new elements in a transitional metamodel MM0. Moreover, MM0 offers a metamodel whose semantic field encompasses those of MM1 and MM2. It allows then an analysis of M1 and M2 seen as two parts of a single model.

6.2 MM0 and OWL

The MM0 metamodel, which represents a means of extending MM1 and MM2 towards the expression of semantics in a more atomic way, will make the use of OWL more efficient by exploiting the transformation that translates MOF based models into ontologies [5]. It is expected that the set of tools supporting OWL will greatly benefit the analysis of each model and of the properties common to both models.

6.3 Automation

The two main tasks requiring a manual input from the developer and therefore being less prone to complete automation are the elaboration of MM0 form MM1 and MM2 and the assignment of weights to metamodel elements. For the first task, the extent of the effort depends on the number of concepts the developer wants to involve in the comparison of both metamodels. On the positive side, the possibility of reusing MM0 for other models derived from MM1 and MM2 can compensate for the effort put in. As for the second task, the developer can take the opportunity to add constraints and requirements such as a choice of correspondence cores or a limitation on elements' membership to correspondence fragments based on some patterns. All the remaining tasks are to be automated.

We are working on a prototype based on the representation of MOF based models in XMI format [8], and relying on algorithms of "reasonable" complexity that can be optimized to handle comparison of large size models.

6.4 Targeted Applications

Below, we mention a non exhaustive list of situations in which the proposed approach can be of benefit.

- Comparison of PIM and PSMs in an MDA development process. The re-use is expected for any couple of technologies.
- Comparison of an application specification model to its implementation model.
- Fusion of a model into another model dealing with a different technology, and written in another language. MM0 will constitute a good starting base.

7 Conclusion and Future Work

Usually, metamodel elements devised to model technologies, comprise elements representing high-level concepts that assist in capturing the complex semantics of these technologies. The approach we have proposed can be seen as a rewriting of models in a simpler language using low-level terms capturing targeted semantics while avoiding semantic overlapping of metamodel elements. We are currently working on a prototype implementing the comparison of models. This prototype will allow us to apply our approach to full blown cases. Though it, we will be able to better illustrate the use of correspondence cores and of weight distributions as parameters guiding the comparison process. We anticipate that this work will open the door to further research endeavours aiming at devising sophisticated, though feasible, methods for the validation of models resulting from transformations.

Another future work consists in defining a metamodel able to describe two distinct technologies (somewhat like MM0), and then integrate a third one, on so on, until obtaining, in the end, a metamodel which is not limited by any of these technologies, but which can describe all their characteristics because of the way it has been built up from their metamodels. Such a metamodel will be an excellent candidate for defining the expression language of a PIM targeting these technologies; the trace of its creation will provide the various transformations towards the targeted PSMs.

References

[1] Abd-Ali, J., El Guemhioui, K.: An MDA-Oriented .NET Metamodel. Ninth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2005), Enschede, The Netherlands (2005) 142-153

[2] Astesiano, E., Bidoit, M., Kirchner, H., Krieg-Brückner, B., Mosses, P., Sannella, D.; Tarlecki, A.: CASL: The Common Algebraic Specification Language. Theoretical Computer Science (2002) 286(2):153-196

[3] Favre, L.: Foundations for MDA-based Forward Engineering. Journal of Object Technology (2005) 4(1):129-153

[4] Fleurey, F., Steel, J., Baudry, B.: Validation in Model-Driven Engineering: Testing Model Transformations. Workshop WS5 at the 7th International Conference on the UML, Lisbon, Portugal (2004). Available @ http://www.metamodel.com/wisme-2004/papers.html

[5] Gaševic, D., Djuric, D., Devedzic, V., Damjanovic, V.: Approaching OWL and MDA Through Technological Spaces. Workshop WS5 at the 7th International Conference on the UML, Lisbon, Portugal (2004). Available @ http://www.metamodel.com/wisme-2004/present/9.pdf

[6] Kleppe, A., Warmer, J. : Do MDA Transformations Preserve Meaning ? An investigation into preserving semantics. First international workshop on Metamodeling for MDA, York, UK (2003)

[7] OMG: Meta Object Facility Specification (MOF)
http://www.omg.org/docs/formal/02-04-03.pdf

[8] OMG: XML Metadata Interchange (XMI) Specification. Available @ http://www.omg.org/docs/fo]rmal/02-01-01.pdf

[9] OWL Web Ontology Language Overview. Available @ http://www.w3.org/TR/2004/REC-owl-features-20040210/

[10] Poernomo, I., Jayaputra, J., Schmidt, h: Timed Probabilistic Constraints over the Distributed Management Task Force Common Information Model. Ninth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2005), Enschede, The Netherlands (2005) 261-272

[11] Thompson, S.:Type Theory and Functional Programming. Addison-Wesley (1991). Available @ http://www.cs.kent.ac.uk/people/staff/sjt/TTFP/

[12] Zamora Zapata, J.P., Bordeleau, F., Corriveau, J.P.: Validation of Platform Specific Models Against Platform Independent Models in the Context of Standard Specifications: A Model Driven Architecture Approach. Proceedings of the 2004 International Conference on Software Engineering Research and Practice (2004)

[13] Zamora Zapata, J.P., Bordeleau, F., Corriveau, J.P., McClean, T.: Challenges and Possible Solutions to Model Compliance in the Context of Model Driven Development. Workshop WS5 at the 7th International Conference on the UML, Portugal (2004). Available @ http://www.metamodel.com/wisme-2004/papers.html