Journal of Universal Computer Science, vol. 15, no. 11 (2009), 2225-2253
submitted: 18/1/09, accepted: 29/5/09, appeared: 1/6/09 © J.UCS

A Flexible Strategy-Based Model Comparison Approach: Bridging the Syntactic and Semantic Gap

Kleinner Oliveira, Karin Breitman
(Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Department of Informatics
Rua Marquês de São Vicente 225, Rio de Janeiro, CEP 22453-900, RJ, Brazil
{kfarias, karin}@inf.puc-rio.br)

Toacy Oliveira
(University of Waterloo
School of Computer Science, Waterloo, ON, Canada
toliveiva@cs.uwaterloo.ca)

Abstract: In this paper we discuss the importance of model comparison as one of the pillars of model-driven development (MDD). We propose an innovative, flexible, model comparison approach, based on the composition of matching strategies. The proposed approach is fully implemented by a match operator that combines syntactical *matching rule*, *synonym dictionary* and *typographic similarity* strategies to a semantic, *ontology-based* strategy. Ontologies are semantically richer, have greater power of expression than UML models and can be formally verified for consistency, thus providing more reliability and accuracy to model comparison. The proposed approach is presented in the format of a workflow that provides clear guidance to users and facilitates the inclusion of new matching strategies and evolution.

Keywords: Model Comparison, Unified Modeling Language, Ontology Alignment, Model Driven Development
Categories: H.3.1, H.3.2, H.3.3, H.3.7, H.5.1

1 Introduction

A significant factor behind the difficulty of developing complex software is the wide conceptual gap between the problem domain and its related computational solution [France, 06], [France, 07]. An attempt to bridge such conceptual gap is the model-driven development [Selic, 03], [Sendall, 03], or MDD, which moves development focus from code to models, in particular those expressed in the Unified Model Language (UML) and its profiles [OMG, 07], [Reddy, 06]. The goal is to manage software at conceptual level, reducing the gap between specifications and code, thus reducing costs and difficulties associated to software development in general.

MDD allows developers and domain specialists to create models to seamlessly represent both static and behavioural concepts found in the problem domain that will be further manipulated and transformed to create the related computational solution (the actual system implementation). For this purpose, two activities are taken into consideration: model transformation and model composition.

Model transformation is comprised of a set of model-to-model (M2M) and model-to-code (M2C) transformation operations. A key step during model-to-model

transformation is model composition, where two or more different models are combined into a single specification. Model composition is a complex task and can be viewed as an operation where a set of activities should be performed over two input models, Ma (the receiving model) and Mb (the merged model), in order to produce an output model Mab. This operation is denoted by the equation: Ma + Mb Mab.

An important step to achieve consistent model composition lays in the ability to compare input model elements. Before composing Ma and Mb, it is necessary to verify the existence of semantic and syntactic overlaps. Overlaps between input models are undesired as they can lead semantic conflicts, misinterpretation and problems in the transformation process. For example, according to the UML metamodel specification [OMG, 07] there should not exist two (or more) model elements, such as two UML classes, with equal names in a same namespace. Therefore, a model composition mechanism attempting to compose UML Class Diagrams should take into account such constraints and avoid creating output models with conflicting names and/or elements with the same semantic value. The model comparison technique is responsible for identifying constraints and defining the correspondences among input model elements, thus making similarity and overlapping explicit.

UML models are often used to describe things that exist the real world is. They are useful to graphically depict a system's structure and behaviour from different viewpoints and at various levels of abstraction. A set of UML models can be used to better manage the description of a system, where each model in the set captures a different aspect of the solution [Sendall, 03] [Ludewig, 03]. Performing model comparison on such models (that take the place of input models Ma and Mb, in our notation) requires a clear understanding of the UML metamodel specification and semantics of both input models.

Matching Ma with Mb means finding a relationship S between a pair of concepts in Ma and Mb in such a way that, if two concepts *r*, in Ma, and *m*, in Mb, are related by *S* (a measure of similarity between concepts *r* and *m*), then *r* and *m* have the same semantic value, as well as syntactic structure (the value that has been assigned to their properties defined in the UML metamodel specification), if the value of *S* is above an empirically or user defined threshold.

Efforts on solving model comparison issues are now gaining momentum at the model driven community. Firstly, model composition should be achieved in a much higher degree of accuracy within the model driven development process. Secondly, flexibility in matching strategies is a much desired non-functional requirement. Additionally, syntactic and semantic aspects should be taken into consideration in a model comparison approach. All such issues are open questions in the literature.

In recent years, several works on model comparison have been proposed, among these are schema matching [Rahm, 01], Web services composition, ontology matching [Euzenat, 07], matching object catalogues [Leme, 08], Statecharts Specifications matching and merging [Nejati, 07], differences between versions of UML diagrams [Ohst, 03], UML model comparison [Kolovos, 06], [Kolovos, 08].We have experimented with a few of the proposed approaches, but found none suitable for usage in UML model comparison. Such approaches ignore important aspects of model comparison that may lead to problems such as: (i) lack of flexibility to determine correspondences among model elements;(ii) poor user interaction; (iii) lack

of focus on model properties; (iv) require a large amount of human effort; (v) most do not scale up to dealing with complex models; (vi) do not take into account the model semantics. Thus, new approaches that overcome these shortcomings are needed.

In this paper we explore the role and the importance of model comparison in model composition; we discuss some of its challenges and propose a flexible model comparison approach. Our main contribution is a match operator, which is responsible for putting in practice the proposed flexible strategy-based model comparison approach that takes into account both syntactic and semantic aspects involved during model comparison. Central to the functioning of the match operator is the capacity of finding precise similarity measurements between pairs of elements in different models. We propose the use of an ontology-based matching strategy that improves the calculation of element similarity while preserving original model semantics. A further contribution of this paper is in the format of a workflow to guide model comparison.

1.1 Motivating Example

We motivated our work with a composition example comprised of two UML profiles, *Tree* (receiving model) and *Topology* (merged model) [Fuentes, 04], each representing a domain-specific modelling language. Figure 1 depicts both profiles. We have chosen UML profiles because they play a central role in the OMG's MDD approach [OMG, 03] and represent an important mechanism to adapt the UML metamodel to a specific-platform domain. The *Tree* profile represents a common hierarchical data structure, used in many computer science applications, while the *Topology* profile represents the connections between the elements of an Information System with a star network topology.

In the Topology profile, there are nodes (represented by stereotype *Node*) connected by links that can be either local (*LocalEdge*), if they connect nodes from the same star with its central node, or remote (*Edge*), if they connect central nodes (*MainNode*) between each other [Fuentes, 04]. Each node is identified by its position (*location*). Each central node has a state (*state*) that defines its availability (its values are defined by enumeration, as per *StateKind* class depicted in Figure 1 a) and b). An end node (*EndNode*) is also identified by its position (*position*). The Tree profile has nodes (represented by stereotype *Node*) connected by links (*Edge*) to other nodes, end nodes (*Leaf*) or root nodes (*root*) that have a state kind (*state*) which defines their availability (its values are defined by enumeration *StateKind*). Each node is determined by its name (*name*) and value (*value*). Moreover, it is possible to perform search operation (*Search*).Before merging Tree and Topology, we should necessarily compare the input profiles in order to merge them efficiently. For this purpose, we need to be able to identify correspondences among UML profile elements in a coherent manner. In other words, identifying relationships between individual elements of the two input profiles is a necessary precondition to the modelling composition phase.

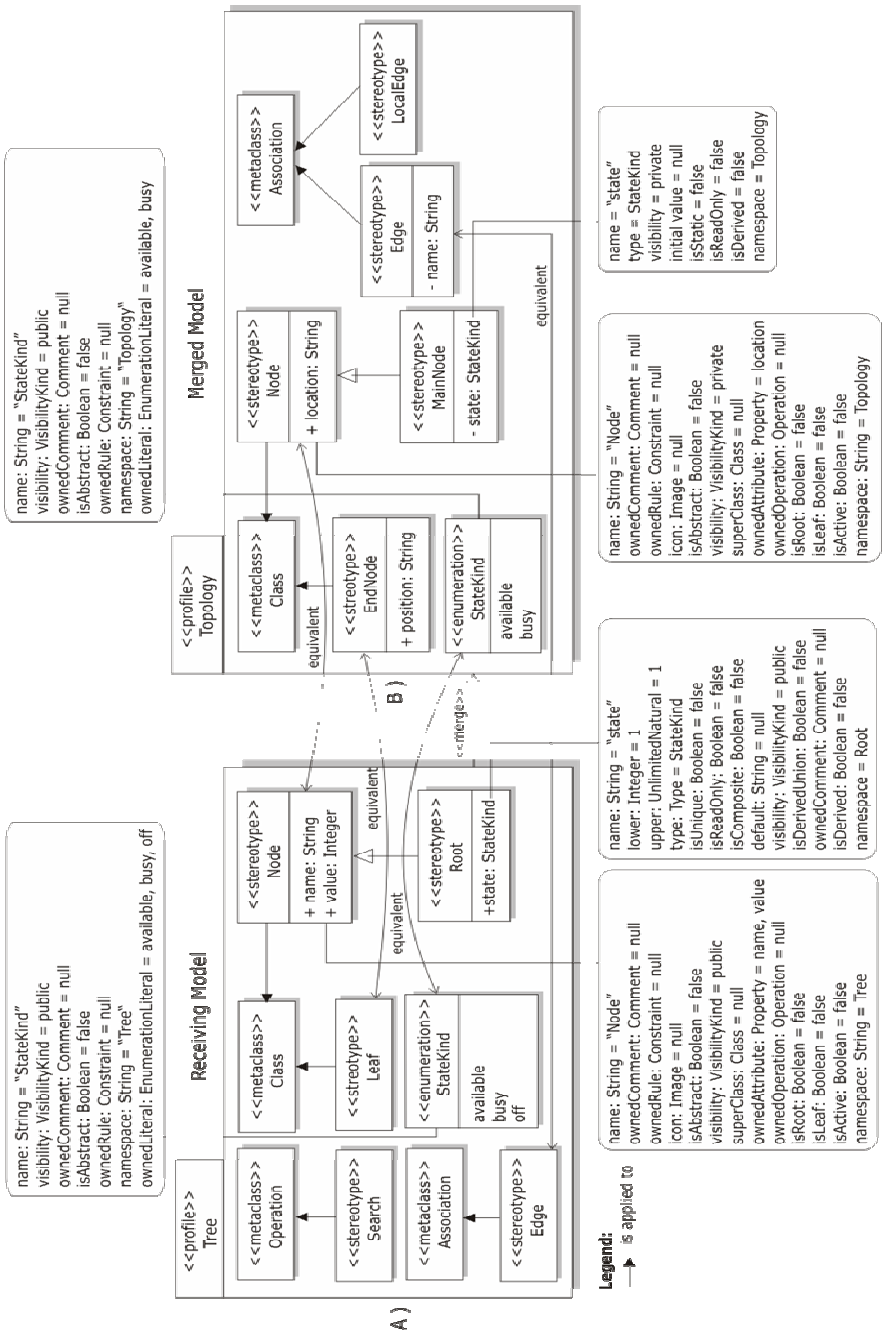

Figure 1: Motivating example A) UML Tree Profile B) UML Topology profile

UML profiles possess a set of properties specified by the UML metamodel specification, which describe what characteristics a stereotype may have. Such properties are key-elements, and should be taken into account during matching.

Figure 1 exemplifies such properties. Note that the stereotype Node has *name* = "Node", *isAbstract* = false, *namespace* = "Tree" and so on. Also observe that, despite the fact that *Tree.Leaf* and *Topology.EndNode* stereotypes have different names, the way they are structurally built, induces the question to whether they could be considered of equal semantic value. The same applies to the *Tree.Root* and *Topology.MainNode* stereotypes. Syntactic similarity poses similar conundrums, observe *Tree.Root* and *Topology.Root*, for instance. They could be considered similar if we take into account the property *name* defined in the UML metamodel. However, when other properties are considered, e.g. *isAbstract* (defines whether a class is abstract or not), *superClass* (the immediate superclass of a class, from which the class inherits), and *ownedAttribute* (the attributes owned by the stereotype) that may not be the case.

1.2 Contributions of this Paper

Putting model comparison in practice involves addressing several of the following model comparison questions: what criteria should we use for identifying correspondences between different models? And how can we quantify these criteria [Nejati, 07]? Should a model comparison approach produce a unique result, one that represents the mappings among elements? What properties of the input models should be considered during matching? What extra information is required so that we can compare models? How can the gap between the syntactic and semantic realms be bridged in matching? If model comparison is as difficult as claimed, then should there exist fundamental limitations that prevent us from accomplishing key aspects of the matching task? Where do such strategies fail, and why? Are published model comparison approaches substantially different? Are differences accidental or essential? These questions are often asked by the would-be practitioners but are not adequately addressed by the growing literature on model comparison.

In this paper we tackle some of these fundamental questions. We propose a flexible model comparison approach based on matching strategies. The proposed approach is implemented by a match operator that further combines them with a range of matching strategies, including the use of typographic similarity, model element signatures, and specially ontology alignment, to augment precise similarity measurements thus obtaining optimal match results. We further propose a set of heuristics, in the format of a workflow, which provides user guidance in specifying the model comparison approach. Our approach is constituent of a UML profile composition mechanism [Oliveira, 08], [Oliveira, 08a], [Oliveira, 07], [Oliveira, 07a] that proved to be an effective and flexible way for specifying correspondences among UML profiles.

The remainder of the paper is organized as follows. In Section 2, we introduce the main concepts and knowledge that are going to be used and discussed throughout the paper. In Section 3, we propose a model comparison approach. We explain and discuss its essential activities, resulting artefacts and the flow of execution. In Section 4, we present the proposed strategy-based approach, in which the match operator plays a central role. Then, continuing in Section 4, we explain how a seamless

integration of syntactic and semantic aspects was weaved to the proposed model comparison approach. In Section 5, we discuss the challenges that researchers face when attempting to put model comparison in practice in the context of MDD, the ontology alignment strategy used in our approach, and the implementations that embody the proposed approach. In Section 6, we contrast the proposed approach with related work. Finally, in Section 7, we present some concluding remarks and future work.

2 Background

Model comparison arises as an essential activity to put the composition phase in practice. It can be viewed as a generic operation that varies from application to application, in which elements from Ma and Mb are compared in different formats, depending on the kind of application in question. For example, matching statechart specifications and different versions of UML diagrams is quite challenging, as the artefacts that are being compared represent properties very differently, so the model comparison strategy must be tailored to suit this specificity.

The UML specification defines and offers practitioners the Profile mechanism that was specified to provide a lightweight extension mechanism to the UML standard. For instance, we can add unspecified semantics to the metamodel, provide a terminology adapted to a particular platform or domain and add information that can be used when transforming a model to another model or directly to code. However, the *Merge* UML built-in composition mechanism is not capable of merging profiles nor comparing input models adequately. So some research questions arise: how can we compare two profile elements? What activities should we perform to match two input models? We may add semantics, that does not exist, in UML metamodel elements, then how can we compare these new elements with new semantics in a flexible manner?

In this paper we propose to capitalize from our previous experience in the project, implementation and integration of ontology based software applications [Breitman, 07], [Leme, 08] to provide a viable solution to UML model comparison. Ontologies are much more expressive than other conceptual models: a controlled vocabulary is a set of terms and definitions, e.g. glossaries and acronyms; taxonomy is a set of terms arranged in a generalization-specialization (parent-child) hierarchy. A taxonomy may or may not define attributes of these terms nor does it specify other relationships between terms, e.g. RosettaNet and ebXML; a relational database schema defines a set of terms through classes, attributes and a limited set of relationships among those classes; an OO software model defines a set of concepts and terms through a hierarchy of classes and attributes and a broad set of binary relationships among classes. Constraints and other behavioral may be specified through methods on the classes (or objects). An ontology can express all of the preceding relationships, models and diagrams as well as, n-ary relations, a rich set of constraints, rules relevant to usage or related processes and other differentiators including negation and disjunction [Goméz-Peréz, 04], [Fensel, 02].

Furthermore ontologies capture knowledge rather than data. Because it is possible to infer new information from previously coded one (with the aid of an inference mechanism), we believe ontologies provide a much more robust conceptual model for

model comparison in the MDA context than restricting ourselves to pure UML models. Table I summarizes the benefits of the adoption of ontology over other conceptual models. Because of the above characteristics, semantic interoperability among ontological models is facilitated [Gómez-Pérez, 04], [Breitman, 07].A few approaches to help dealing with ontology integration include merging, alignment, mapping [Noy, 03] and integration. Merging ontologies results in an unique model containing the sum of concepts from the original ontologies, without indication of its provenance.

Benefits of the adoption of ontology
1. Ontologies are semantically richer (greater expression power than taxonomies, entity relationships or OO models).
2. Conceptual knowledge is maintained through complex and accurate representations above and beyond hierarchical approaches.
3. Ontologies are formal - OWL DL ontologies map directly to Description Logic (a dialect of first order logics).
4. Formal ontologies in the OWL DL standard can be verified/classified with the aid of Inference Mechanisms, e.g. RACER and FaCT: (i) consistency checks; (ii) classification; and (iii) new information discovery.
5. OWL ontologies use a XML/RDF syntax that allows them to be automatically manipulated and understood by most resources on the Internet.
6. Capture and represent finely grained knowledge.
7. Ontologies can be used to reduce ambiguity so as to provide a model over which information can be freely shared and acted upon by autonomic managers.
8. Ontologies are modular, reusable and code independent - ontology driven applications are specified separately from the ontology itself. Changes to the ontology should not impact the code or vice-versa.

Table 1: Benefits of using ontology to represent input models

Aligning ontologies is the identification of the links that hold between concepts from two different inputs. Those links provide the shared semantics of both representations. Alignment is usually done in pairs. The result of the alignment of two input ontologies can be presented either in the format of an intermediate representation (third “aligned” ontology) or as an addition to the markup of the original ontologies. Mapping between two ontologies results in a formal representation that contains expressions that link concepts from one ontology to the second. This result is of particular interest to the UML model comparison approach proposed in this paper, for it provides formal, unambiguous, accurate and precise similarity measurements for pairs of model elements, while preserving their original semantics. In the next section we propose a model comparison approach that makes use of one such ontology based matching strategy.

3 Model Comparison Process

There is little agreement on requirements, activities and steps that should be followed in order to accomplish reliable model comparison, and even less on good practices to avoid errors during matching. Several works have been proposed to tackle the problems found in model comparison, but none of them, as yet, was defined as standard [Ohst, 03], [Kolovos, 06]. According to the Object Management Group (OMG), the UML built-in model comparison approach does not present a task flow to help compare UML models, neither it does present good documentation, much less a definition of how model comparison should be performed [OMG, 07]. In what follows we succinctly describe the model comparison approach proposed in [Oliveira, 08b] where we define a series of activities, and an accompanying workflow to guide users. The focus of this paper is on the comparison stage, explored in further detail in section 4.

3.1 Overview

We illustrate the proposed approach in Figure 2. Note that it is comprised of three phases, namely initial, comparison and merge. The initial phase begins when the match operator, detailed in the following section, receives the two input models. The activity performed in this phase is the identification and analysis of the input models. The goal of the comparison phase is to define what input model elements are similar. It is initially performed by analysing both input models and defining a signature for every model element type. Then the match operator, with the aid of a set of previously chosen matching strategies, calculates the similarity degree (S) for every input model element in Ma in relation to model elements present in Mb. The major contribution of this paper lies in this step, in which we identify good matches, while preserving the semantics of the input models, combining an ontology-based matching strategy with the merely syntactical ones proposed in [Oliveira, 08a].

On the following step the match operator determines pairs of similar model elements based on an empirically or user defined threshold (t). The phase is finished as soon as the matching models, no matching models and matching description are specified. The final stage is merging the models; however this activity is outside the scope of this paper.

In what follows we detail each phase and related activities of the proposed model comparison approach.

3.2 The Initial Phase

During this phase, the identification and analysis of the input models occur. The goal is to identify the types of elements in both models, e.g. Stereotypes, Classes, and Associations, so that the match operator, detailed in the next section, is able to determine whether it is able to compare them or not. Using the motivation example depicted in Figure 1, the elements Stereotypes (*Tree.Node* and*Topology.Node*) and Association (*Tree.Edge* and *Topology.Edge*) are identified and grouped according to their types. Model elements are separated in types to reduce the problem spaces of the matching strategies used in the next phase, comparison.

3.3 Comparison Phase

This phase is the core of the proposed model comparison process. It is comprised of five distinct steps, summarized in Table 2 and described as follows.

In the first step the domain expert defines a signature for every model element type that can be defined as input model. In the second step the user specifies what matching strategy should be followed. Once the matching strategy has been determined, the match operator will put the matching in practice according to the strategy chosen. In our approach, we provide a choice among three matching strategies: default, partial and complete, described in more detail in the next section. The output of this activity is a matching strategy specification that is going to be used next. Following is the stage in which the degree of similarity (*S*) between pairs of elements from the two different input models is calculated. As stated earlier, identify relations between individual elements of multiple models is a necessary precondition to put model composition in practice. The match operator receives as input the match strategy specification and combines the synonym dictionary, Ngram algorithm [Manning, 99], matching rules, and similarity measurement from the ontology alignment strategy implemented by CATO [Breitman, 05] to calculate the similarity degree between pairs of elements.

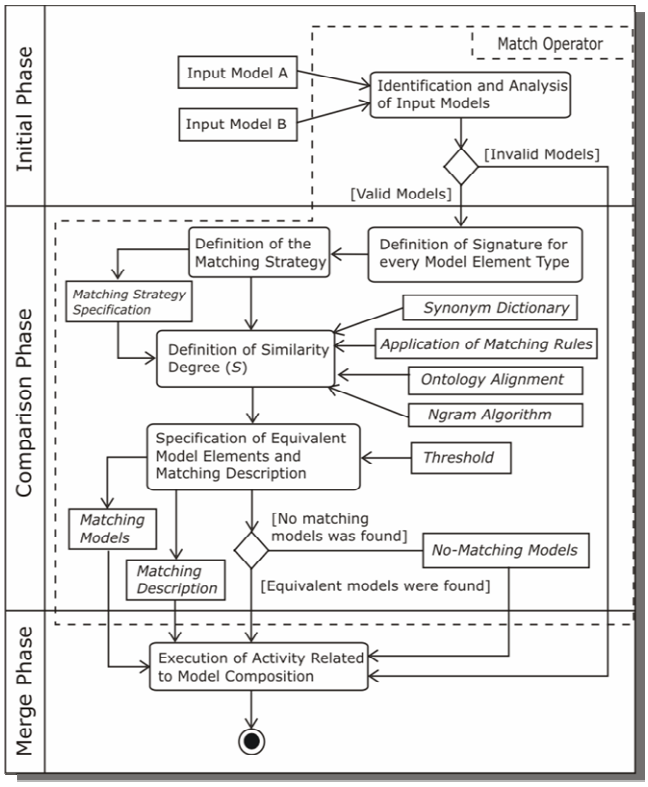

Figure 2: Proposed model comparison approach (evolved from [Oliveira, 08b])

The next step focuses on the specification of the equivalent model elements and matching description. For this purpose, at the beginning of model comparison, the user determines a similarity threshold. Every pair of elements whose similarity degree is above the threshold is considered equivalent. The output generated by the match operator consists of two artefacts, the first are the matching models, i.e., a set of models that are considered equivalent, and the second is a matching description, i.e., a description of how models, in the first artefact, establish a equivalence relationship, in other words, it defines all matching pairs. This activity is explicit in the model comparison approach because of our work was initially conducted with a focus in of model composition [Oliveira, 08], [Oliveira, 08a], [Oliveira, 07], [Oliveira, 07a].

Comparison Phase
Activities
1. Definition of a signature for every model element type
2. Definition of the matching strategy.
3. Computation of the similarity degree between pairs of input model elements using the ontology based strategy implemented by CATO.
4. Specification of equivalent elements and matching description
5. Performing activities related to model composition

Table 2: Activities comprised in the comparison phase.

4 Proposed Strategy-Based Model Comparison Approach

Once having established a motivating example and succinctly described the model comparison approach, we proceed to proposing a flexible model comparison approach based on matching strategies. During the building of the model comparison approach, a constant concern was to provide a seamless integration of syntactic and semantic aspects. Thus, throughout this section, we are going to discuss how to put this in practice.

The model comparison approach proposed by Oliveira et al anticipates a choice among three pre defined strategies (i) default, (ii) partial, and (iii) complete (see [Oliveira, 08], [Oliveira, 08b] for more details). However, the approach is extensible and allows for the addition of new strategies. The central goal of the model comparison approach is offering matching flexibility, that is, allowing for the use of different strategies and making it possible to have different descriptions of similarity based on the strategy choice. This feature is particularly interesting if we take into account that different matching strategies work better in some specific domains than others. This problem is very well known to the natural language processing community, where the application domain is a deciding factor in choosing matching strategies [Manning, 99], [Wang, 04].

In our particular case, the art of employing UML models relies in assigning suitable values to previously defined metamodel properties. Following this reasoning, and recognizing the fundamental role played by UML properties in model comparison, useful strategies are those capable of specifying a set of closely related

properties and evaluating these during matching (comparison phase, see Figure 2). Observing the motivating example, we can see some properties of Stereotype (e.g., *name*, *ownedComment*, and *ownedRule*) and Enumeration (e.g., *name*, *ownedLiteral*, and *visibility*) that have been previously defined. Therefore, defining a model comparison strategy is to specify what properties are going to be considered during matching.

For example, the *Default Matching Strategy* takes into account only the property "*name"* that is defined in every UML model. Thus, when the stereotypes *Tree.Root* and *Topology.Node* are compared, following the Default Matching Strategy, their names and the names of their attributes, e.g. *Tree.Root.state* and *Topology.Node.location*, will be considered. The success and feasibility of applying strategies in matching is not surprising if we think carefully about and putting UML model's properties at the heart of their definition.

A *match operator* was created to serve as the responsible mechanism for combining and putting the matching strategies in practice. Using the input models and the matching strategy specification, the match operator verifies the equivalence degree among the input model elements and, according to a threshold, specifies the matching model. The operator is described as follows.

4.1 The Match Operator

The match operator is a mechanism whose goal is to find correspondences among input model elements using a model comparison strategy. The operator takes into consideration both the syntactical and semantic aspects of the input models throughout matching.

On the syntactical hand, it performs both lexical and structural comparisons in order to determine if model elements in different input models should be considered syntactically equivalent. On the semantic hand, it takes into consideration the expertise of domain specialists from the domain in discussion as much as domain ontologies in a seamless way. The operator analyses whether concepts that have been explicitly defined by specialists as synonymous can be found in the input models. Moreover, it accomplishes lexical and structural comparisons in order to determine if domain concepts in different ontologies should be considered semantically equivalent.

The operator combines the *model signature* and *typographic similarity* strategies to tackle syntactical aspects; and *synonym dictionary* and *ontology alignment* to tackle semantic aspects of the model comparison approach. All strategies are combined so as to determine the equivalence degree (S) between model elements. We detail the above mentioned strategies as follows.

4.1.1 Synonym Dictionary

With a *synonym dictionary* it is possible to identify mapping among domain concepts that have equal semantic values. The great benefit of using synonym dictionaries is to pave the way for the domain specialists to explicitly apply their domain expertise in matching. We denote by $D(r,m)$ [0,1] the degree of similarity between receiving (r) and merged(m) model elements, e.g., *Tree.Node* and *Topology.MainNode*, respectively. $D(r,m)$ returns 0, whether *r* and *m* are synonyms, otherwise it returns 1. *D* is calculated for every possible pair of (r,m). Initially, every pair (r,m) of input

model elements is not assumed to be synonymous, then D(r,m) = 0 for every pair of (r,m). For instance, according to synonym dictionary (Table 3) the stereotypes *Tree.Leaf* and *Topology.EndNode*, depicted in motivating example (Figure 1), represent the same concepts, therefore *D*(Leaf,EndNode) = 1.

Name	**Synonym**
leaf	EndNode, FinalNode
edge	Border, Limit, Margin
search	Research, Searching, Query

Table 3: Example of Synonym Dictionary

4.1.2 Typographic Similarity

The goal of typographic similarity is to determine T(r,m) [0..1] for every possible pair of receiving (r) and merged (m) model elements. The N-gram algorithm [Manning, 99] is applied to assign a similarity value in [0..1] to every possible pairs of (r,m). These pairs are determined by cartesian product of (R×M), where R and M are the set of receiving and merged model elements, respectively. The result of R×M is the matrix shown in Table 4. This algorithm yields a similarity degree to a pair of strings based on counting the number of their identical substrings of length N (we use N = 2).

4.1.3 Model Signature

The *signature* is defined in terms of model element syntactic properties, where a syntactic property of a model element defines its structure. Such syntactic properties are illustrated in the motivating example, e.g. *Tree.Node.isRoot* and *Tree.StateKind.visibility*. The signature is a collection of values assigned to a subset of syntactic properties in a model element's metamodel class. For example, *isAbstract* is a syntactic property defined in the metamodel class called Class. If an instance of a Class is an abstract class then *isAbstract* = true for the class, otherwise the instance is a concrete class, *isAbstract* = false. The set of syntactic properties used to determine a profile element's signature is called *signature type*, as defined in [Reddy, 06].

Three type of signature were defined: (i) *complete signature,* which consists of all syntactic properties associated with a model element; (ii) *partial signature*, which is made up a range of syntactic properties; and (iii) default signature, which is composed only by *name* properties. The signatures can be structured in comparison levels organized hierarchically. For instance, in Figure 1, a possible definition of levels for the stereotype *Tree.Node* would be: *Tree.*Node (name) (level 2), with *Tree.Node.name* and *Tree.Node.value* (tagged values) (level 1). Every model element type should have a signature. The similarity degree based on signature *M* between receiving (r) and merged (m) model element is represented by *M*(r,m), where and $0 \leq M \leq 1$ and $M \in \mathbb{R}_+$. It is defined by calculating the weighted average among the arithmetic average of the levels (see Equation 1):

$$\mathcal{M} = \frac{\sum_{i=1}^{n} p_i \cdot \left[\sum_{j=1}^{k} \frac{\varphi_{i,j}}{k} \right]}{\sum_{i=1}^{n} p_i} \rightarrow [0..1] \qquad (1)$$

- n is the number of levels employed to compare the model elements, where $n \geq 1$ and $n \in \mathbb{N}^*$. For example, we defined three levels to compare stereotypes from input models. The first level having the property *name: String* only. The second one having the *ownedAttribute: Property*. The third one having the *ownedOperation: Operation.*
- p_i represents the weight, being $p_i = i$, where $i \geq 1$ and $i \in \mathbb{N}^*$; k expresses the number of elements in each level, where $k \geq 1$ and $k \in \mathbb{N}^*$. (e.g. *Tree.Node* has two properties, as these properties represent a level, so $k = 2$);
- $\varphi_{i,j}$ (i and j represent the level and item of model elements that are being compared, respectively) is used to denote if an item of receiving model element (e.g., *name:String* in *Tree.Node*) is equivalent to another item of merged model element. Matching rules were designed to function as a boolean variable, in the sense that the result of their application falls into two value ranges: 1 if the rule is satisfied, and 0 if the rule is not satisfied. For example, when we compare the *Tree.Root* and *Topology.MainNode* stereotypes, $\varphi_{2,1} = 0$, applying the matching rule MR1, and, $\varphi_{1,1} = 1$, applying the matching rule MR3.

4.1.3.1 Verification Using Matching Rules

In order to check if a pair of input model elements is equivalent, we defined matching rules. The match operator is responsible to execute these matching rules and, according to the result of this execution, it defines consequently the value of $\varphi_{i,j}$, which was specified earlier. For every model elements there is a matching rule to check whether they are equivalent or not. This checking is based on their signature. If a matching rule fails, then the models are not equivalent ($\varphi_{i,j} = 0$). Otherwise, models are equivalent ($\varphi_{i,j} = 1$). The matching rules verify whether the input model element properties have the same values, and for each matching strategy is defined a set of matching rules according to respective signature type of the strategy.

There are three kinds of matching rules: (i) *default matching rules* are a set of rules that compare models based on only their name, using the default signature type; (ii) *partial matching rules* are also a set of rules that compare models based on a number of syntactic properties of the models, using the partial signature type; (iii) *complete matching rules* are also a set of rules that compare models based on their syntactic properties, using the complete signature type. Thus, the match operator makes use of these rules to implement the default, partial and complete matching strategies, respectively. For example, the match operator makes use of the default matching strategy (hence using default matching rules) to produce the similarity table

depicted in Table 4. We present a short description of the default matching rules applied in the motivation example, as what follows:

MR1. Stereotype matching rule:
MatchStereotype(Stereotype rcv, Stereotype mrgd)
rcv.name = mrgd.name AND
MatchAttribute(rcv, mrgd) AND MatchOperation(rcv, mrgd)

MR2. Association matching rule:
MatchAssociation(Association rcv, Association mrgd)
(rcv.name = mrgd.name) AND (rcv.memberEnds = mrgd.memberEnds)

MR3. Attribute matching rule:
MatchAttribute(Stereotype rcv, Stereotype mrgd)
(rcv.ownedAttribute.name = mrgd.ownedAttribute.name)
AND (rcv.ownedAttribute.TypedElement = mrgd.ownedAttribute.TypedElement)

MR4. Operation matching rule:
MatchOperation(Stereotype rcv, Stereotype mrgd)
(rcv.ownedOperation.name = mrgd.ownedOperation.name)
AND (rcv.ownedOperation.ownedParameter.length =
mrgd.ownedOperation.ownedParameter.length) AND
($\forall$ x(rcv.ownedOperation.ownedParameter[x]=
mrgd.ownedOperation.ownedParameter[x])

MR5. Enumeration matching rule:
MatchEnumeration(Enumeration rcv, Enumeration mrgd)
rcv.name = mrgd.name AND
MatchEnumerationLiteral(Enumeration rcv, Enumeration mrgd)

MR6. Enumeration Literal matching rule:
MatchEnumerationLiteral(Enumeration rcv, Enumeration mrgd) $\forall$
x(rcv.ownedLiteral.name[x] = mrgd.ownedOperation.name[x])

4.1.4 Ontology Alignment

In this paper we adapt from an existing ontology integration strategy as an innovative means to obtain more precise similarity measurements. Such measurement is represented by O, where $0 \leq O \leq 1$ and $O \in \mathbb{R}_+$. Initially designed to provide mappings between two input ontologies, the ontology alignment strategy proposed in [Felicissimo, 04] is implemented by the CATO tool [Breitman, 05] (see Figure 3).

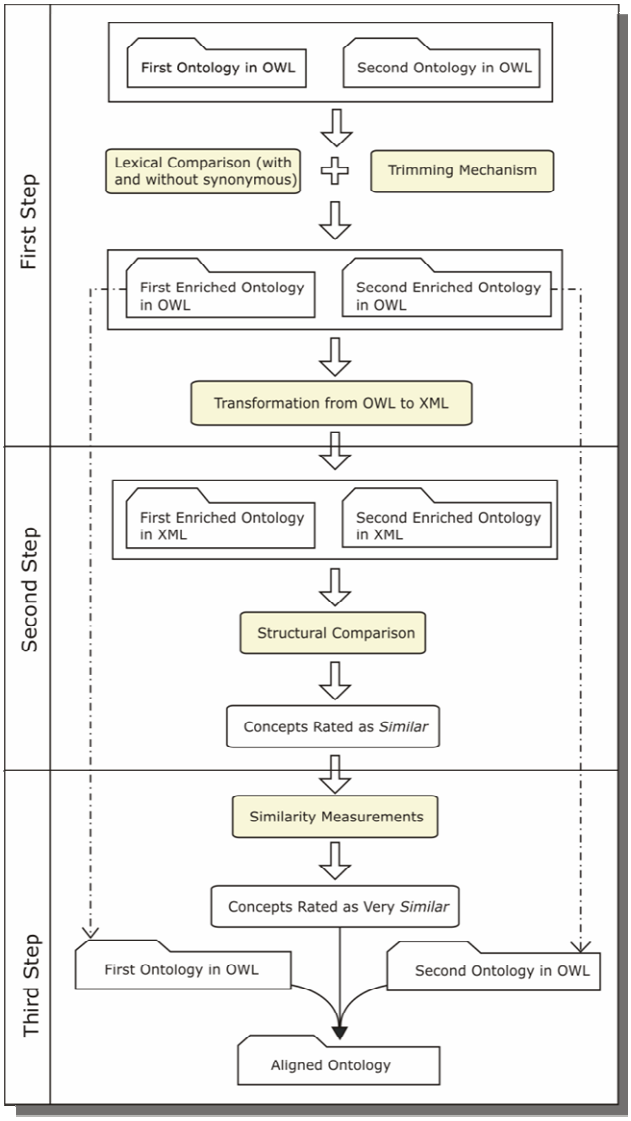

Figure 3: CATO ontology alignment strategy

CATO takes as input any two ontologies written in W3C recommended standard OWL. It was fully implemented in JAVA and uses a specific API (Application Programming Interface) that deals with ontologies, JENA [Jena, 09]. It performs both lexical and structural comparisons in order to determine if concepts in different ontologies should be considered semantically equivalent. It is based in a refinement approach, broken into three successive steps, detailed in what follows.

4.1.4.1 First Step: Lexical Comparison

The goal of this step is to identify lexically equivalent concepts. We assume that lexically equivalent concepts are also semantically equivalent in the domain of discourse under consideration, an assumption that is not always warranted. Each concept label in the first ontology is compared to every concept label present in the second one, using lexical similarity as the criteria. Figure 4 shows the compared ontologies in our study. Filters are used to normalize the labels to a canonical format: (i) if the concept is a noun, the canonical format is the singular masculine declination; (ii) if the concept they represent is a verb, the canonical format is its infinitive.

Besides using the label itself, synonyms are also used. The use of synonyms enriches the comparison phase (see Figure 2) because it provides more refined information. Lexical similarity alone is not enough to assume that concepts are semantically compatible. We also investigate whether their ancestors share lexical similarity. For example, the motivating example concepts *Leaf* and *EndNode* were identified as synonyms in our database. It is important to note that the alignment strategy in this step is restricted to concepts and instances of the ontology. We are not considering properties at this time. A concept instance is represented by a pair name and namespace in OWL. As a result of the first stage of the proposed strategy, the original ontologies are enriched with links that relate concepts identified as lexically equivalent.

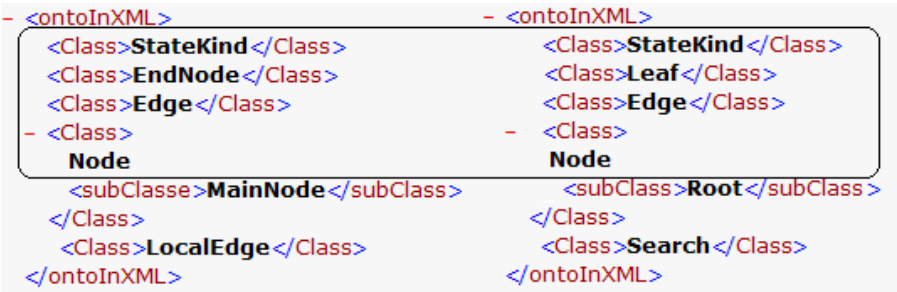

Figure 4: Compared ontologies

4.1.4.2 Second Step: Structural Comparison Using TreeDiff

Comparison at this stage is based on the subsumption relationship that holds among ontology concepts. Ontology properties and restrictions are not taken into consideration. Our approach is thus more restricted than the one proposed in [Noy, 03], that analyses the ontologies as graphs, regarding both taxonomic and non taxonomic relationships among concepts. Because we only consider lexical and structural relationships in our analysis, we are able to make use of well-known tree comparison algorithms. We are currently using the TreeDiff [Wang, 98] implementation available at [Bergmann, 02] (see Figure 5).

Our choice was based on its ability to identify structural similarities between trees in reasonable time. The goal of the TreeDiff algorithm is to identify the largest

common substructure between trees, described using the DOM (Document Object Model) model. This algorithm was first proposed to help detect the steps, including renaming, removing and addition of tree nodes, necessary to migrate from one tree to another (both trees are the inputs to the algorithm).

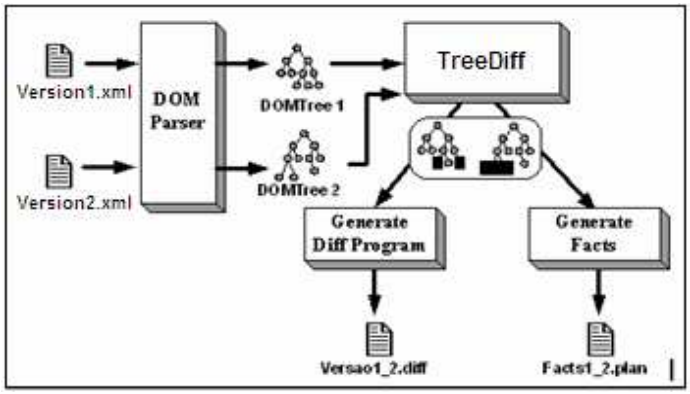

Figure 5: The TreeDiff algorithm's entries and exits [Bergmann, 02]

The result of the Tree Diff algorithm is the detection of concept equivalence groups. They are represented as subtrees of the enriched ontologies. Concepts that belong to such groups are compared in order to identify if lexically equivalent pairs can also be identified among the ancestors and descendants of the original pair. Differently from the first step, where we based our analysis and compared concepts that were directly related to one another, we are now considering the structural vicinity of concepts. Every concept in the equivalence group is investigated in order to determine lexically equivalent pairs, number of matching sons, number of synonymous concepts in the sub-trees, available from the previous step, and ancestor equivalence.

However, not a single equivalence group has been detected in our case study is illustrated in Figure 4. Note that the ontologies in the case study possess two concepts named as *Root* and *MainNode*. Additionally, in both cases, their super-concept is labelled with *Node*. The equivalence group is not thus identified by CATO. All concepts in the equivalence group are then compared. The concepts *Root* and *MainNode*, despite being differently labeled in both ontologies, have a lexically equivalent super-class (*Node* in Ontology *Tree* and *Node* in Ontology Topology), and are thus classified as no equivalent. Besides they act on the similarity measurement of *Node*.

4.1.4.3 Third Step: Fine Adjustments based on Similarity Measurements

The third and last step is based on similarity measurements. Concepts are rated as very similar or little similar based on pre-defined similarity thresholds. We only align concepts that were both classified as lexically equivalent in the second step, and thus rated very similar. Thus the similarity measurement is the deciding factor responsible

for fine tuning our strategy. We adapted the similarity measurement strategies proposed in [Bergmann, 02], [Maedche, 01].

This is the case of concepts *Node*, *StateKind*, *Edge*, from the case study. Those concepts were rated equivalent during the second step. Their similarity level is calculated in the present step. Figure 6 depicts the results. The final ontology, shown in Figure 7, provides a common understanding of the semantics represented by the two input ontologies. As long-term goal, this representation can now be accessed by model comparison operator searching for information or knowledge to compare UML models. In this paper, we make use of the similarity measurements, represented by *O*, as cited previously.

```
Similarities Level:
StateKind -> StateKind    *** Similarity Level: 100.0%
Node -> Node              *** Similarity Level: 50.0%
Edge -> Edge              *** Similarity Level: 100.0%
```

Figure 6: Similarity percentages for concepts in the equivalence group illustrated in Figure 4

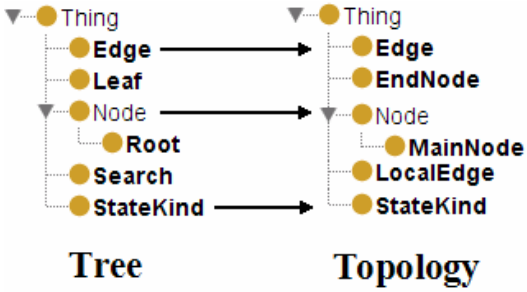

Figure 7: Aligned concepts between ontologies

4.2 Calculating the Similarity Degree Between the Input Model Elements

We denote by *S* the degree of similarity between receiving (r) and merged (m) model elements. For defining the similarity degree, it is necessary to combine the partial similarity degrees described in previous sections. For this purpose, it is calculated the average of *D, T, M*, and *O* as shown in Equation 2. If *D* = 1, then T also assumes value 1 and contrariwise.

$$S = \frac{D + T + M + O}{D + 3} \rightarrow [0..1] \qquad (2)$$

Where:

D – Synonym Dictionary similarity degree, calculated as indicated in section 4.1.1. Note that if D = 1, then T also assumes value 1 and contrariwise.

T – Typographic Similarity, calculated as indicated in section 4.1.2

M – Model Signature Similarity, calculated as indicated in section 4.1.3

O – Ontology Alignment Similarity, calculated as indicated in section 4.1.4

Based on Equation 2, we calculate the similarity degree of every *Tree* element in relation to *Topology* elements. Table 4 shows the matching results. To produce a correspondence relation between the two models, we set a threshold (t = 0.7). This is an arbitrary choice, sufficient, however for the purposes of this paper. So, pairs of model elements with similarity degree above threshold are considered equivalent. In short, if S(r,m) > t, then r and m are equivalents. In Table 4, we point out the similarity degree above threshold and define the profile elements are equivalent, as follows: (*Tree.Node*, *Topology.Node*), (*Tree.Edge*, *Topology.Edge*), (*Tree.Leaf*, *Topology.EndNode*) and (*Tree.StateKind*, *Topology.StateKind*).

Tree Profile \ Topology Profile	Node	MainNode	Edge	LocalEdge	EndNode	StateKind
Node	0,74	0,15	0	0,05	0	0
Root	0	0,03	0	0	0	0
Edge	0	0	1	0,14	0	0
Search	0	0	0	0	0	0
Leaf	0	0	0	0	0,75	0
StateKind	0	0	0	0	0	0,97

• similarity degree above the threshold (t = 0,7)

Table 4: Similarity degree among the motivating example's profile elements

5 Discussion

5.1 Challenges in Model Comparison

To the best of our knowledge, the need for comparing models in a flexible manner neither have been pointed out nor even proposed by current model comparison approaches in the context of the model composition mechanisms or any other work discussed in previous sections. This fact shows the pioneer side of this work. Based on earlier works [Oliveira, 07], [Oliveira, 07a], [Oliveira, 08], [Oliveira, 08a] and relevant approach have been studied and discussed in the last section, we have observed and concluded that the major challenges, that researches face when attempting to put model comparison into practice in the context of MDD, can be grouped into the following categories:

- The *domain-specific model comparison challenge*: Such challenge arises from concerns associated with providing DSMLs for creating and using domain specific models in the MDD vision. For example, the UML supports

two forms of extensions: (1) using profiles to define UML variants and (2) associating particular semantics to specified semantic variation points [OMG, 07], [France, 07]. Hence, a challenge is how to develop model comparison tailoring support strategies that take into consideration the semantics plugged into UML semantic variation points and possible specializations of the UML metamodel profiles.

- The *abstraction level challenge*: Once the MDD vision manipulates models in different abstraction levels, how should the model comparison approach provide support for matching models expressed in different abstraction-levels? This challenge addresses problems related to the understanding and evolution of model comparison approaches across different modeling languages.
- The *semantic and properties challenge*: As models have semantic values associated to them, any pair of elements with the same name and equal semantic value can automatically be assumed to form a match. However, what should be done if the pairs have different semantic values or different properties? To illustrate, imagine two input UML classes with same name, however one is abstract and the other is concrete. While the pair of classes may still be considered a match, there is a conformance mismatch between them.

5.2 Alignment Strategy

For the sake of efficiency, the ontology approach implemented by CATO only takes into consideration syntactical information, i.e., lexical and structural equivalence, in the proposed alignment strategy. However, this limitation of the strategy can be overcome by the adaptation of the second step to take into consideration other ontology primitives, such as properties (the strategy could work with graphs instead of trees) and axioms. For sure this adaptation will increase the total computation time because of the added complexity.

In the current implementation the strategy depicted in Figure 3 is fully automated, sequential and does not allow for the possibility of user feedback. Because every step of the strategy refines the previous one, more precise results can be achieved if manual, user feedback is allowed.

Furthermore, the CATO alignment strategy is very conservative, in that it discards doubtful matches to preserve reliability. The worst case scenario in terms of completeness is not being able to align any pair of elements. This happens when the input ontologies are from disjoint domains. The worst case scenario in terms of inconsistency is aligning two concepts that have identical names, but are semantically different. This would only happen if, and only if, both shared identical names and possessed a great deal of structural similarity, i.e., lexically equivalent concepts (synonyms) as descendants and/or ancestors.

5.3 Implementation issues

After the theoretical description of our approach, we now need to deploy them. One needs a proof of concept. For this purpose, we are going to discuss three implementations that embody the proposed approach: (i) a tool for UML profiles

composition, that is, the MoCoTo (Model Composition Tool) implemented and described in [Oliveira, 08a]; (ii) a MoCoTo Software Product Line; (iii) a Lightweight Ontology Alignment Tool, namely, CATO [Breitman, 05], [Felicissimo, 04]. With these implementations we automated and put in practice our matching strategies. Moreover, they have shown concrete evidence so as to satisfy our initial claims. The infrastructure fulfilling the implementations will be carefully described in what follows.

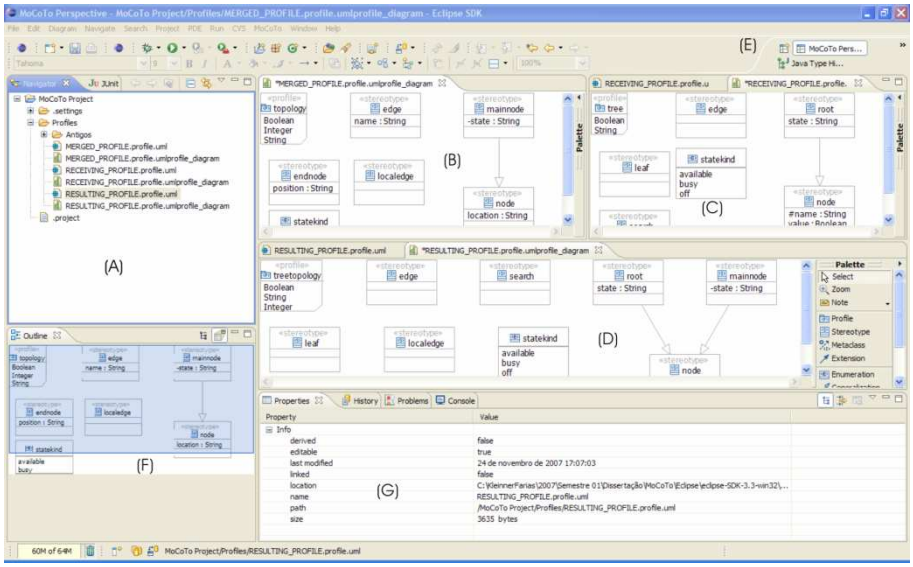

Figure 8: MoCoTo overview

5.3.1 Model Composition Tool (MoCoTo)

MoCoTo is an Eclipse Plug-in which permits a seamless integration with Eclipse Platform. It provides functionalities for users work with model composition and model comparison in the Eclipse SDK. The goal of MoCoTo is to automatically compute Mab (the resulting model) from the matching models and matching description produced by the match operator following the steps described in previous sections. For this purpose, MoCoTo makes use of multiple Eclipse modeling technologies, e.g. EMF [EMF, 09], UML2 [UML2, 09], GEF [GEF, 09], UML2 tool [UML2Tool, 09], in order to allow the modelers and developers to implicitly compare UML profiles and explicitly merge them. MoCoTo ties together these technologies in a such way that makes it easy to use, even to users with little or no Java or XML coding experience. The use of such technologies helped us focus on matching, for they make matching transparent. For example, UML2 API reads and filters information from the tags of files written in XML and transforms it to an abstract data model in which input model elements can be manipulated as objects. EMF, UML2 and UML2 tool ease to create the UML input models and to implement the matching strategies and the matching rules.

Figure 8 depicts an overview of the MoCoTo. For each created project, its source folders, files and referenced libraries are organized in a tree structure being possible to open and browse the contents of them (A). The input models illustrated in the motivating example are found at (B) and (C). The output model produced using our approach is shown at (D). The concept of perspective defined in the Eclipse Platform was employed in our tool. The MoCoTo perspective defines the initial set and layout of views in the Workbench window and provides a set of functionality aimed at accomplishing a set of tasks or works related to modeling and model composition (E). Moreover, the tool has outline view, which shows the overview of the model being manipulated (F), and property view, which allows the model's properties are edited and checked (G).

5.3.2 MoCoTo Software Product Line (MoCoTo SPL)

Scope, commonality, and variability analysis gives software engineers a systematic way of thinking about and identifying the product family they are creating and manipulating [Coplien, 98]. With this in mind, we create a Software Product Line (SPL) to derive model composition tool from MoCoTo discussed previously. In Figure 9, MoCoTo SPL is represented by a feature diagram, in which its commonalities and variabilities are mapped. Commonality represents the kernel of the SPL (the obligation feature) whereas variability represents the optional and alternative parts of the SPL (the optional and alternative features).

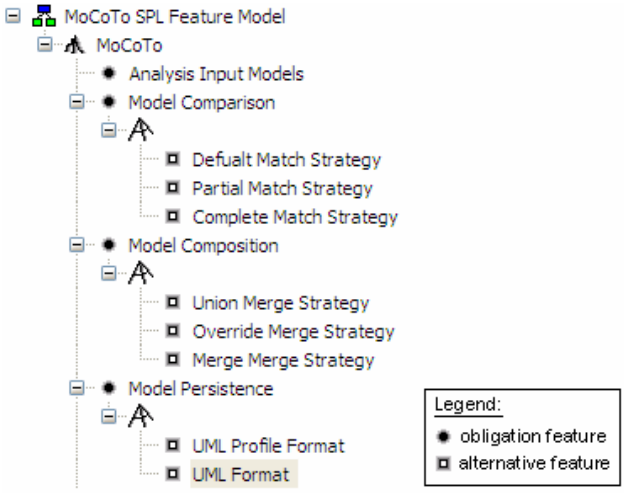

Figure 9: Feature model of the MoCoTo SPL

Four obligation features compose the kernel of our MoCoTo SPL, among them we can point out the *model comparison feature,* the focus of this discussion. The proposed match operator is the great responsible for putting the model comparison feature in practice. The default, partial, and complete matching strategy were defined as alternative features (see Figure 9) and implemented using the aspect oriented

programming; they are encapsulated in an *aspect*. When a new product is generated, it is necessary, first, to weave the algorithms relating to each strategy into an object, namely *Matching Strategy*. The operator has a reference to this object. Then it can access and make use of the strategy available. During the development some Eclipse technologies were employed. For example, EMF [EMF, 09], UML2 [UML2, 09], GEF [GEF, 09], and UML2 tool [UML2Tool, 09]. Additionally, we use AspectJ in order to make use of the resource presented in the aspect oriented programming.

Finally, MoCoTo SPL helps developers and modellers create a design that contributes to reuse and ease of change, predict how a design might fail or succeed as they evolve, and identify opportunities for reusing and automating the creation of new products.

5.3.3 A Lightweight Ontology Alignment Tool (CATO)

CATO [Breitman, 05], [Felicissimo, 04] was fully implemented in Java and relies on the use of the JENA API. The use of the API helped us focus on the alignment process, for it made ontology manipulation transparent. JENA reads and filters information from the tags of files written in an ontology language and transforms it to an abstract data model in which ontological concepts can be manipulated as objects.

During the construction of CATO some adjustments to the algorithms had to be made. In particular, the refinement of the algorithms used in the manipulation of equivalence groups (step 2 of the strategy) made a great impact in the alignment results. We experimented with two manipulation algorithms for the equivalence groups. In the first implementation, we alphabetically ordered the sons of each node of the equivalence group sub-tree. In the second implementation, we maintained the original order in which the concepts appeared in the ontology. For pairs of ontologies that made use of identical labels, the use of the alphabetically ordered structural comparison files brought significantly better results. This was due to the fact that, after the alphabetical sort, the concepts present in an equivalence group will be more closely located in the structure. However, ontologies that make use of identical labels are rarely the case in practice. We tested the performance of both files on ontologies that had few identically labelled concepts. The ordered file did not bring differences in the results and there were some cases when it made the results worse than using the unordered file.

The TreeDiff algorithm, used in the second step of the strategy, is unidirectional in the sense that its goal is to determine the transformation needed to go from the first input tree to the second input tree. We are currently experimenting with double runs of the algorithm in which we invert the order of the input (today we are using the biggest ontology as the first input). When compared, the results present some differences. We believe we can refine the results from this algorithm by providing the combination of the two runs. Future plans include continuing validation of the approach by experimentation and the elaboration of more case studies.

6 Related Work

Model comparison approaches are widely applied in different domains and contexts, and play a central role in several open issues in real world applications such as: model

composition, schema integration, schema evolution and migration, merging of source code, ontology integration, matching class diagram, differencing and merging of architectural views, application evolution, database integration, differences between XML documents, and differences between versions of UML diagrams.

Semantics interoperability among ontologies and database schema matching, for instance, have been in the research agenda of sematics web researchers and knowledge engineers for a while now [Euzenat, 07], [Bernstein07, Wang04, Rahm01]. A few approaches to help deal with the ontology integration problem have been proposed. The most prominent ones are: merging [Noy, 99], alignment [Noy, 99], [Noy, 03], [Ehrig, 07], mapping [Noy, 03], Castano04] and integration [Pinto, 99]. A good overview about ontology integration is also given in the [Bruijin, 06].For example, regarding particularly ontology integration, it pays attention in representing semantic interoperability among ontologies and has been in the research agenda of knowledge engineers for a while now.

Thus, previous research works have proposed many techniques to tackle the inherent problems related to matching, and achieved an automation degree in matching operation for specific application domains. With this in mind, an extensive investigation on related works is necessary before an own innovative approach is developed. Such works have been carefully selected to depict a wide range of model comparison application scenarios. They show many facets of model comparison and therefore are capable of covering different application and user views.

In what follows we make a comparative analysis of related work. First, we give an overview of the approaches relevant to our work in adding flexibility to the model comparison process. Figure 10 summarizes our findings. An ideal real world model comparison application would be a combination of the strengths of each approach, rather than one in particular.

Model Composition Semantics. S. Clarke [Clarke, 01] introduces composition semantics for UML class diagrams. The approach defines a new design construct, called composition relationship that supports the specification of how design models should be composed. With this composition relationship it is possible to: (i) identify and specify overlapping and non-overlapping concepts; (ii) specify how models should be integrated, and how conflicts in equivalent elements are reconciled. The identification of the overlapping parts is based on the name of the input models alone; it is a weakness of the approach.

Model Composition Directives. [Reddy, 06] presents a model composition technique that relies on signature matching, in which model elements are merged if their signatures are correspondent. By contrast, the match operator in our work makes use of a synonym dictionary, typographic similarity and ontology alignment in addition to the model signature, thus providing more reliable similarity degree values.

Assessment Criteria

Approaches	Guidance	Strategies	Typography	Name	Structural	Semantic	Rules	Operator
Model Composition Semantics				X				
Model Composition directives				X	X			
Ontologies Merging			X	X				
Package Merge				X			X	
Epsilon Merging Language				X	X			
Difference Between Models				X				

Figure 10: Comparison of related approaches

Ontologies Merging. While part of our work focuses on fully automated merging of ontologies, there are several semi-automatic ontology merging tools available. The GLUE system [Doan, 03] makes use of multiple learning strategies to help find mappings between two ontologies. Given any two ontologies and their instances, the system is able to find nodes that are similar, given a pre defined similarity measurement. It is an automated tool that feeds its result to a semantic interoperability tool that can interpret and make use of its results. Iprompt provides guidance to the ontology merge process by describing the sequence of steps and helping identify possible inconsistencies and potential problems. AnchorPROMPT [Noy, 03] an ontology alignment tool, automatically identifies semantically similar terms. It uses a set of anchors (pairs of terms) as input and treats the ontology as a directed graph. In this graph, the nodes are the ontology classes and the links its properties. It makes use of similarity measurements and equivalence groups to help detect similar terms. The Chimaera environment [McGuinness, 00] provides a tool that merges ontologies based on their structural relationships. Instead of investigating terms that are directly related to one another, Chimaera uses the super and subclass relationships that hold in concept hierarchy to find possible matches. Their implementation is based in Ontolingua editor [Farquhar, 96].

Package Merge. It is the composition mechanism of the UML [OMG, 07] and is defined by matching rules, constraints and transformation (the merge rules). The major application is in the implementation of the UML compliance levels. In principle, their matching rules are similar to match used by our match operator. However, the OMG proposed matching rules are expressed in natural language, so matching takes into consideration the name of the models only. Moreover, the definition of Package Merge is incomplete, ambiguous and inconsistent.

Epsilon Merging Language. EML [Kolovos, 06] [Kolovos, 08] is a metamodel agnostic language for expressing model composition. It includes a model comparison and a model transformation language as subsets. Model comparison is based only in syntactic criteria. By contrast, matching, in our approach is founded in both

syntactical and semantic aspects which should be carefully taken in consideration throughout the model comparison approach.

Difference between Models. Ohst et al. propose an approach to detect and visualize differences between versions of UML documents, e.g., class or object diagrams. It produces a unified document which contains the common and specific parts of the two input documents, where the specific parts are highlighted [Ohst, 03]. While our approach tackles a range of very difficult problems related to dealing with comparison of semantics values in a flexible manner, theirs is primarily concerned with the comparison and manipulation of models within the same domain and with assumed equal semantic values; there is no flexibility in the authors' comparison process.

7 Concluding Remarks and Future Work

If models are seen as primary development and composition artefacts in model driven engineering, then software designers naturally become concerned with how they are manipulated in a key activity: model comparison. In order to be considered for use in mainstream software development, model comparison techniques should be supplemented with flexibility and taking into consideration both syntactical and semantic aspects of the input models.

In this paper we discussed the importance of model comparison as one of the pillars of the model composition approach. We discussed some of its problems and made explicit some of the theoretical and implementation challenges involved. We proposed an innovative flexible model comparison approach, based on the composition of syntactical and semantic match strategies, implemented and coordinated by a match operator.

The possibility of combining different matching strategies assures overall better performance and reliability in the comparison phase. Depicted in the format of an intelligible workflow (see Figure 2), it provides clear guidance to users and facilitates the inclusion of new matching strategies and evolution.

Nevertheless, our approach presents some limitations that should be further investigated. When models are defined, it is possible to associate them semantics constraints. These constraints should be considered when performing model composition, so that the specified semantics is not challenged. Our approach as of yet, is not able to deal with the issue of comparing such constraints. We expect to enhance the functionality of the match operator by creating new match strategies and improving the matching rules to deal with constraints in the near future.

Even though our approach has been fully implemented and integrated to a profile composition mechanism, further empirical studies are necessary to validate the approach in real world design settings, to verify its performance levels and its applicability in different application domains. Finally, we note that improvement in model comparison approaches is paramount to the evolution of model engineering and its adoption by our industrial peers.

We hope that the issues and challenges outlined throughout the paper encourage researchers to cope with the matching conceptual models problem, thus fostering a new generation of methods, tools and strategies to support the task.

References

[Bergmann, 02] Bergmann, U.: Evolution of Scenarios through a Tracking Mechanism Based on Transformations (in Portuguese), PhD Thesis of the Department of Informatics of PUC-Rio, 2002.

[Bernstein, 07] Bernstein, P., and Melnik, S.: Model management 2.0: manipulating richer mappings, In: Proc. 2007 ACM SIGMOD International Conference on Management of Data, pp. 1–12. ACM Press New York, NY, USA, 2007.

[Breitman, 05] Breitman, K., Felicissimo, C., and Casanova, M.: CATO - A Lightweight Ontology Alignment Tool, In CAiSE Short Paper Proceedings, 2005.

[Breitman, 07] Breitman, K., Casanova, M. and Truszkowski, W.: Semantic Web: Concepts, Technologies and Applications, Springer Verlag, 2007.

[Bruijin, 06] Bruijn, J., Ehrig, M., Feier, C., Martíns-Recuerda, F., Scharffe, F., and Weiten, M.: Ontology Mediation, Merging, and Aligning, Semantic Web Technologies, pp. 95-113. 2006.

[Castano, 04] Castano, S., Ferrara, A., Montanelli, S., and Racca, G.: Semantic Information Interoperability in Open Networked Systems". In: Proc. International Conference on Semantics of a Networked World (ICSNW), in cooperation with ACM SIGMOD, Paris, France, 2004.

[Clarke, 01] Clarke, S.: Composition of Object-Oriented Software Design Models, Ph.D. dissertation, School of Computer Applications, Dublin City University, Dublin, Irland, January 2001.

[Coplien, 98] Coplien, J., Hoffman, D., and Weiss, D.: Commonality and Variability in Software Engineering, IEEE Software, pp. 37-45, November/ December, 1998.

[Doan, 03] Doan, A., et. al.: Learning to match ontologies on the Semantic Web. In: The VLDB Journal — International Journal on Very Large Data Bases, Volume 12, Issue 4, 2003. ISSN: 1066-8888. pp. 303-319, 2003.

[Ehrig, 07] Ehrig, M.: Ontology Alignment – Bridging the Semantic Gap, Springer, 2007.

[EMF, 09] Eclipse Modeling Framework (EMF) Project, Available at: http://www.eclipse.org/modeling/emf/, Accessed on January, 2009.

[Euzenat, 07] Euzenat, J., Shvaiko, P.: Ontology matching, Springer, Springer-Verlag, Berlin Heidelberg (DE), 2007.

[Farquhar, 96] Farquhar, A., Fikes, R., Rice, J.: The Ontolingua Server a Tool for Collaborative Ontology Construction, In Proceedings of the Tenth Knowledge Acquisition for Knowledge Base Systems Workshop, Banff, Canada, 1996.

[Felicissimo, 04] Felicissimo, C.: Semantic Interioperability on Web: a Strategy to Taxonomical Alignment of Ontology (in Portuguese), , Master's thesis, Department of Informatics, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil, August, 2004.

[Fensel, 02] Fensel, D.: Ontology Based Knowledge Management, IEEE Computer, November, pp. 56-59, 2002.

[France, 06] France, R., Ghosh S., Dinh Trong, T.: Model Driven Development Using UML 2.0: Promises and Pitfalls, IEEE Computer Society, vol. 39, no. 2, pp. 59–66, February 2006.

[France, 07] France, R., and Rumpe, B.: Model-Driven Development of Complex Software: A Research Roadmap, In Proc. Future of Software Engineering (FOSE'07), co-located with International Conference on Software Engineering (ICSE'07), Minnesota, EUA, May 2007, pp. 37–54.

[Fuentes, 04] Fuentes, L., and Moreno, A.: An Introduction to UML Profiles, In The European Journal for the Informatics Professional, vol. 5, no. 2, pp. 6–13, April 2004.

[GEF, 09] Graphical Editing Framework (GEF) Project, Available at: http://www.eclipse.org/gef/, Accessed on January, 2009.

[Gómez-Pérez, 04] Gómez-Pérez, A., Fernadéz-Peréz, M., and Corcho, O.: Ontological Engineering, Springer Verlag, 2004.

[Jena, 09] Jena, the Semantic Web Framework, Available at: http://jena.sourceforge.net/, Accessed on January, 2009.

[Kolovos, 08] Kolovos, D.: Epsilon Merging Language Project Page (Epsilon), 2008, http://www.eclipse.org/gmt/epsilon/.

[Kolovos, 06] Kolovos, D., Paige, R., and Polack, F.: Model Comparison: a Foundation for Model Composition and Model Transformation Testing, In International Workshop on Global Integrated Model Management, New York, NY, USA: ACM Press, pp. 13–20, 2006.

[Leme, 08] Leme, L., Brauner, D., Breitman, K., Casanova, M., and Gazola, A.: Matching Object Cataloques, Inovations in System Software Engineering, Springer, 2008.

[Ludewig, 03] Ludewig, J,: Models in Software Engineering: an Introduction, Journal on Software and Systems Modeling, vol. 2, no. 1, pp.5–14, March 2003.

[Maedche, 01] Maedche, A., and Staab, S.: Comparing Ontologies Similarity Measures and a Comparison Study, Institute AIFB, University of Karlsruhe, Internal Report, 2001.

[Manning, 99] Manning, C., Schutze, H.: Foundations of Statistical Natural Language Processing, ISBN 978-0262133609, MIT Press, 1999.

[McGuinness, 00] McGuinness, D., Fikes, R., Rice, J., and Wilder, S.: The Chimaera Ontology Environment, In Proceedings of the 17th National Conference on Artificial Intelligence (AAAI), 2000.

[Nejati, 07] Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S., and Zave, P.: Matching and Merging of Statecharts Specifications, In ICSE'07, Minnesota, EUA, pp. 54–64, May 2007.

[Noy, 03] Noy, F., Musen, A.: The PROMPT Suite: Interactive Tools For Ontology Merging And Mapping, International Journal of Human-Computer Studies, 2003.

[Noy, 99] Noy, F., Musen, A.: SMART: Automated Support for Ontology Merging and Alignment, In. Proc. Workshop on Knowledge Acquisition, Modeling, and Management, Banff, Alberta, Canada, 1999.

[Ohst, 03] Ohst, D., Welle, M., and Kelter, U.: Differences between Versions of UML Diagrams, In 9th European Software Engineering Conference, ACM Press, pp. 227–236, 2003.

[Oliveira, 08] Oliveira, K., and Oliveira, T.: Model Comparison – A Strategy-Based Approach, In 20th International Conference on Software Engineering and Knowledge Engineering (SEKE'2008), San Francisco, USA, 2008.

[Oliveira, 08a] Oliveira, K.: Composition of UML Profiles (in Portuguese), Master's thesis, Informatics Faculty, Pontifical Catholic University of Rio Grande do Sul, Porto Alebre, Brazil, February 2008.

[Oliveira, 08b] Oliveira, K., Silva, M., Oliveira, T., and Alencar, P.: A Flexible Approach to Compare UML Models (in Portuguese), In II Brazilian Symposium on Software Components, Architectures, and Reuse, Porto Alegre, 2008.

[Oliveira, 07] Oliveira, K., and Oliveira, T.: Composition of UML Profiles, In Proc. Workshop Thesis and Dissertation on Software Engineering, co-located with Brazilian Symposium on Software Engineering, pp. 25-30, 2007.

[Oliveira, 07a] Oliveira, K., and Oliveira, T.: A Guidance for Model Composition, In. Proc. International Conference on Software Engineering Advances (ICSEA'07), pp. 27–32, August 2007.

[OMG, 07] Unified Modeling Language: Infrastructure version 2.1, Object Management Group, February 2007.

[OMG, 03] Object Management Group (OMG): MDA Guide (version 1.0.1), http://www.omg.org/docs/omg/03-06-01.pdf, 2003.

[Pinto, 99] Pinto, S., Goméz-Peréz, A., and Martins, J.: Some Issues on Ontology Integration. In: Workshop on Ontologies and Problems Solving Methods: Lessons Learned and Future Trends. Proceedings of the Workshop on Ontologies and Problem Solving Methods: Lessons Learned and Future Trends (IJCAI99), 1999.

[Rahm, 01] Rahm, E., Bernstein, P.: A Survey of Approaches to Automatic Schema Matching, VLDB Journal, vol. 10, no. 4, pp. 334-350, 2001.

[Reddy, 06] Reddy, Y., France, R., Straw, G., Bieman, N., Song, E., and Georg, G.: Directives for Composing Aspect-Oriented Design Class Models, Transactions of Aspect-Oriented Software Development, vol. 1, no. 1, pp. 75–105, 2006.

[Selic, 03] Selic, B.: The Pragmatics of Model-Driven Development, IEEE Software, vol. 20, no. 5, pp. 19–25, September/October 2003.

[Sendall, 03] Sendall, S., and Kozaczynski, W.: Model Transformation: The Heart and Soul of Model-Driven Software Development, IEEE Software, vol. 20, no. 5, pp. 42–45, September/Octocber 2003.

[UML2, 09] Unified Modeling Language (UML2) Project, Available at: http://www. eclipse.org/ modeling/mdt/?project=uml2, Accessed on January, 2009.

[UML2Tool, 09] Unified Modeling Language (UML2) Tool Project, Available at: http://www. eclipse.org/modeling/mdt/?project=uml2tools, Accessed on January, 2009.

[Wang, 98] Wang, J.: An Algorithm for Finding the Largest Approximately Common Substructures of Two Trees, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, no. 8, pp. 889-895, 1998.

[Wang, 04] Wang, J., Wen, J., Lochovsky, F., and Ma, W.: Instance-based schema matching for web databases by domain-specific query probing, In Proc.13th Int'l. Conf. on Very Large Data Bases, pages 408–419, Toronto, Canada, 2004.